HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Sebastian Flemig

Konzeption, Entwurf und prototypische Implementierung einer Simulations-Lösung für Test und Inbetriebnahme von IP-basierten Sicherheitssystemen mit proprietären Kommunikationsschnittstellen

2010

Fakultät **Mathematik / Naturwissenschaften / Informatik**

BACHELORARBEIT

Konzeption, Entwurf und prototypische Implementierung einer Simulations-Lösung für Test und Inbetriebnahme von IP-basierten Sicherheitssystemen mit proprietären Kommunikationsschnittstellen

Autor:
Sebastian Flemig

Studiengang:
Informatik

Seminargruppe:
IF06w1-B

Erstprüfer:
Prof. Dr. Wilfried Schubert

Zweitprüfer:
Dr. Mike Bergmann

Mittweida, 2010

Bibliografische Angaben

Flemig, Sebastian: Konzeption, Entwurf und prototypische Implementierung einer Simulations-Lösung für Test und Inbetriebnahme von IP-basierten Sicherheitssystemen mit proprietären Kommunikationsschnittstellen, 91 Seiten, 21 Abbildungen, Hochschule Mittweida (FH), Fakultät Mathematik / Naturwissenschaften / Informatik

Bachelorarbeit, 2010

Referat

Ziel dieser Bachelorarbeit ist es, eine Software zur Virtualisierung von IP-basierten Sicherheitssystemen zu konzeptionieren und prototypisch zu implementieren. Dazu wird zuerst die ursprüngliche Situation in der Firma beleuchtet und ein Konzept für eine Virtualisierungssoftware entwickelt. Neben Betrachtungen zur Wahl der Programmiersprache und in welcher Art Daten über das Netzwerk übertragen werden sollen, muss die GUI designed werden und die Art der Datenhaltung entschieden werden. Auch bieten sich im Firmenumfeld viele, dem Gelegenheitsprogrammierer meist unbekannte, Techniken und Anwendungen an, wie das Testen des Softwareprototyps mit PHPUnit und Hudson, die Datenbankanbindung mit PDO oder das Bearbeiten bereits geladener Webseiteninhalte mit AJAX.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Listingverzeichnis

1 Einleitung

1.1 Motivation

Unsere Umwelt wird zunehmend digitalisiert. Computer und Handys sind allgegenwärtig und selbst die Handys können meist schon als Computer bezeichnet werden. Onlinedienste wie beispielsweise Onlinebanking oder Nachrichtenportale erleben eine immer größere Verbreitung. Warum sollte dieser Trend also vor der Sicherheitstechnik halt machen?
Die Zeiten, in denen Schlösser ausschließlich mechanisch über einen Schlüssel aufgesperrt werden konnten, sind schon lange vorbei. Mit der Entwicklung der jeweiligen Technik wurde diese auch mehr oder minder erfolgreich in der Sicherheitstechnik eingesetzt. Techniken, die eine Person (oder auch Gegenstände oder Tiere) automatisch identizifierten, werden Auto-ID genannt. Dazu zählen Barcodes, OCR, Magnetstreifen, biometrische Identifikation (Retina, Fingerabdruck, Sprache etc.), Chipkarten, RFID und optische Datenträger. [Vgl. Ker06, S. 15]
All diese Auto-ID-Techniken haben Vor- und Nachteile, sind teils besonders teuer in der Anschaffung (Biometrie), benötigen direkten Kontakt, sodass ein Schloss nicht im Vorbeigehen geöffnet werden kann (Chip- und Magnetstreifenkarten, Fingerabdruck) oder sind sehr unpraktikabel und letztendlich auch unsicher, da sie mit sehr einfachen Mitteln kopierbar sind (Barcode, OCR).
RFID hat mit allen Verfahren lediglich einen gemeinsamen Nachteil: es muss erst die jeweilige Infrastruktur geschaffen werden aus Lese- und Schreibgeräten, sowie Sicherheitstechnik, die mit diesen zusammenarbeiten kann. Die Technik kristallisiert sich als Alleskönner heraus, denn je nach Einsatzzweck kann die Infrastruktur anders geplant werden (andere Sender, andere Empfänger). Dadurch können manche Vorteile auf Kosten anderer – in dieser Hinsicht eventuell akzeptablerer – Nachteile erkauft werden. [Vgl. Ker06, S. 40] Somit kann diese Technologie sowohl im Supermarkt als Diebstahlschutz, als auch zur Wiedererkennung von Wildtieren oder als Zutrittskontrolle in Sicherheitsbereiche eingesetzt werden. [Vgl. TT10, S. 3]
Diese Vielseitigkeit, alle möglichen gestalterischen Formen vom einfachen fingernagelgroßen Aufkleber bis hin zum massiven Plastikschlüsselanhänger oder auch die Form einer gewöhnlichen Chipkarte anzunehmen, prädestiniert RFID für Sicherheitsanwendungen aller Art und Sicherheitsstufen.

1.2 Ziele der Arbeit

Diese Arbeit beschäftigt sich mit der Konzeption einer Virtualisierungssoftware für eine Sicherheitslösung unter Wahrung einer Trennung zwischen der Darstellung (GUI) und der Programmlogik, nachfolgend auch Controller genannt.
Um diese Software effizient zu gestalten und die Zeit für die prototypische Umsetzung

so gering wie möglich zu halten, müssen verschiedene softwaretechnische Verfahren angewandt werden. Dafür unterteilt sich die Arbeit in drei fließend ineinander übergehende Phasen.

1. In der *Konzeptionsphase* wird die zu erstellende Software geplant und zu erreichende Ziele abgesteckt. Dies macht die Erstellung eines Pflichtenheftes, im unternehmerischen Umfeld Softwarespezifikation genannt, erforderlich. Für das Pflichtenheft wird die Gliederung nach Balzert [Vgl. Bal09, S. 491] bevorzugt, da diese Quasi-Standard an der Hochschule Mittweida (FH) ist.
2. In der nachfolgenden *Entwurfsphase* müssen konzipierte Funktionen, Klassen und Daten effizient dargestellt und verknüpft werden. Dies macht die Erstellung von Klassen- und „Use Case"-Diagrammen notwendig, um eine übersichtliche Durchführung der dritten Phase zu gewährleisten. Besonders komplexe Sachverhalte werden mit Aktivitätsdiagrammen dargestellt.
3. In der letzten Phase, der *prototypischen Implementierung* werden die Ziele und Vorstellungen, die in der Konzeptions- und Entwurfsphase festgelegt worden sind, prototypisch umgesetzt. Am Ende dieser Phase soll eine lauffähige Software stehen, die sich in Zukunft problemlos benutzen und erweitern lassen soll.

Diese Phasen machen es erforderlich, sich mit Objektorientierung und dem MVC-Modell (siehe Unterabschnitt 3.2.2) auseinanderzusetzen. Es muss die Entscheidung getroffen werden, welche Programmiersprache zur Implementierung verwendet wird, sowie in welcher Form Daten gespeichert werden sollen. Für die Netzwerkfähigkeit muss die Eignung verschiedener Datenübertragungsarten betrachtet werden. Weiterhin muss der im Laufe der Arbeit entstehende Programmcode durchgehend dokumentiert werden und es muss mittels Tests jederzeit die Funktion der Software überprüft werden können.

1.3 Aufbau der Arbeit

Das Ziel dieser Arbeit ist eine lauffähige prototypische Implementierung der Virtualisierungssoftware.
In diesem Kapitel erfolgt deshalb die Einleitung mit der Vorstellung der eigenen Motivation sowie die Ziele der Arbeit.
Im zweiten Kapitel werden die Aufgabenstellung analysiert und eventuell auftretende Probleme betrachtet und Entscheidungen abgeleitet.
Theoretische Hintergründe werden im dritten Kapitel erläutert. Mit diesen sind die Überlegungen hinter der prototypischen Softwareentwicklung, sowie deren Voraussetzungen verständlicher. Das Kapitel enthält eine Abwägung von PHP gegenüber Java, sowie inwiefern sich eine dieser Programmiersprachen besser zum Programmieren von Services, sowie zur Trennung von Programm- und Darstellungslogik eignet. Zudem wird erörtert wie die Software Daten versendet.
Das vierte Kapitel behandelt konzeptionelle Festlegungen zur GUI, sowie Entscheidungen

bezüglich der Verwendung der Datenbanksysteme und der Skriptsteuerung. So umfasst es z.B. die Argumente für und wider MySQL oder SQLite.
Im fünften Kapitel werden einige ausgewählte programmiertechnisch interessante Details aufgegriffen, die eine besondere Herausforderung darstellten oder sehr elegant gelöst werden konnten. Eine Weiterverwendung dieser Programmteile in anderen Projekten ist sehr wahrscheinlich.
Das letzte Kapitel umfasst eine kritische Sicht auf die erreichten Ergebnisse sowie einen Ausblick über zukünftige Weiterentwicklungen der Software und welche Rolle diese im Unternehmen spielen wird. Zusätzlich wird erläutert, welche Möglichkeiten für weitere Bachelor- oder Diplomarbeiten sich aus dieser Arbeit heraus ergeben können.
Im Anhang A findet sich anschließend die Softwarespezifikation nach Balzert [Vgl. Bal09, S. 491] für diese Virtualisierungssoftware. Auf ausführliche Funktionsbeschreibungen wird verzichtet. Stattdessen werden User Stories verwendet, die eine wesentlich kürzere und auch für Laien verständliche Beschreibung der Funktionen ermöglichen. Des Weiteren sind in Anhang B einige Bildschirmfotografien des letzten Standes der prototypisch implementierten Software enthalten.

2 Problemanalyse und Aufgabenstellung

2.1 Ist-Zustand

Um ein Endgerät, z.B. ein Gerät (nachfolgend auch Box genannt) mit Schubfächern und RFID-Leser, welches bei richtiger Identifikation ein Fach öffnet, richtig betreiben zu können, ist eine Software im Backend erforderlich, die auf einem Server läuft. Diese Software lernt das Gerät an, teilt ihm mit, welche RFID-Transponder welche Fächer öffnen dürfen und bekommt Rückmeldungen beim Öffnen von Fächern oder bei Alarmauslösung (z.B. bei Aufbruch oder wenn ein Fach blockiert ist).
Bei verschiedenen Vorgängen sendet die Box an den Server Protokolle. In entgegengesetzter Richtung läuft Datenverkehr zum Öffnen von Fächern, Setzen von Berechtigungen oder Anlernen von Fächern. Beim Anlernprozess wird der Box mitgeteilt, welcher Transponder (auch Dallas genannt) in einem Fach liegen sollte.
Dieser Datenverkehr ist zwingend und muss stattfinden, auch wenn neue Funktionen in die Sicherheitssoftware integriert werden sollen und diese getestet werden müssen. Beim Test neuer Funktionen oder bei Änderung bestehender Funktionen muss deshalb eine Box an den PC angeschlossen werden, der für die Entwicklung verwendet wird.
Eine derartige Box wiegt ab 50 kg und ist nur in begrenzter Stückzahl für den Softwaretest verfügbar. Sie muss deshalb häufig ab- und an einem anderen Ort aufgebaut werden, was durch das hohe Gewicht mühselig ist. Auch bei der Softwarevorführung beim Kunden muss eine solche Box mitgenommen werden. Die logistischen Anforderungen sind hoch. In Abbildung 2.1 ist solch eine Box abgebildet.

Abbildung 2.1: Beispiel einer Fachanlage mit Kartenleser und Touchscreen

2.2 Aufgabenstellung

Es soll eine Softwarelösung entworfen und prototypisch implementiert werden, die es ermöglicht, automatische und manuelle Tests von Software, die ursprünglicherweise eine physische Box benötigen, durchzuführen. Die Software soll alle Systemkommunikation und Protokolle des Hardwarecontrollers (C165) implementieren und somit die Virtualisierung einer Box möglich machen.

2.3 Anforderungen

Die Software soll ein grafisches Interface bereitstellen, welches eine Übersicht über derzeitige Depot- und Systemzustände gewährt. Dieses Interface soll anschaulich genug sein, um auch bei Vorführungen beim Kunden eingesetzt werden zu können.
Weiterhin sollte eine Steuerung der Software per Skript möglich sein. Anstatt alle Einstellungen von Hand über die GUI vorzunehmen, soll es möglich sein die Software automatisch über dieses Skript zu konfigurieren. Ist die Software über ein Skript konfiguriert worden, soll es unbeaufsichtigt betrieben werden können. Alle Aktionen und Reaktionen auf eintreffende Datenpakete sollen protokolliert werden.
Es ist eine Trennung zwischen Programmlogik (Controller) und Darstellung (View) einzuhalten. Zudem soll die Software auf Windows PCs einsetzbar sein ab Windows XP. Sie soll allein durch Kopieren (und dann eventuell nötige Konfiguration) auf einem Rechner installiert werden können.
Während der prototypischen Implementierung und auch beim späteren Einsatz der Software soll die fehlerfreie Funktionsfähigkeit der Software getestet werden können. Für diese Tests bieten sich Unit Tests an.

2.4 Einsatzzweck und Zielgruppe der Software

Die Software soll hauptsächlich zwei Zwecke erfüllen:

1. Bei *Softwaretests* soll sie das physische Vorhandensein einer Box überflüssig machen.
2. Wenn eine *Softwarepräsentation* beim Kunden durchgeführt werden soll, bei der dem Kunden „seine“ Software mit angepasster Oberfläche (z.b. an die Firmen-CI) gezeigt wird, soll die Software die Darstellung testweise geöffneter und geschlossener Fächer, sowie Einlagerung und Entnahme von Transpondern visualisieren. Auch das macht die Mitnahme einer physischen Box unnötig.

Als Zielgruppe kommen ausschließlich Mitarbeiter der Firma KEMAS in Betracht. Es ist nicht vorgesehen, dass die Software vom Kunden bedient wird.

3 Theoretische Hintergründe

3.1 Zusammenspiel zwischen Box und Software

3.1.1 Erste Version der Boxarchitektur

Um eine Box virtualisieren zu können, ist ein Verständnis der Arbeitsweise der physischen Boxen notwendig.
In frühen Versionen bestanden diese aus Depots, Kartenlesern, Tastenfeldern und eventuell einem LC-Display – alles direkt über einen Bus mit dem C165 verbunden (siehe Abbildung 3.1). Dieser Chip kümmerte sich um die Prüfung der Zustände der Depots sowie um das Senden und Empfangen von Protokollen, z.B. bei Öffnung eines Depots.

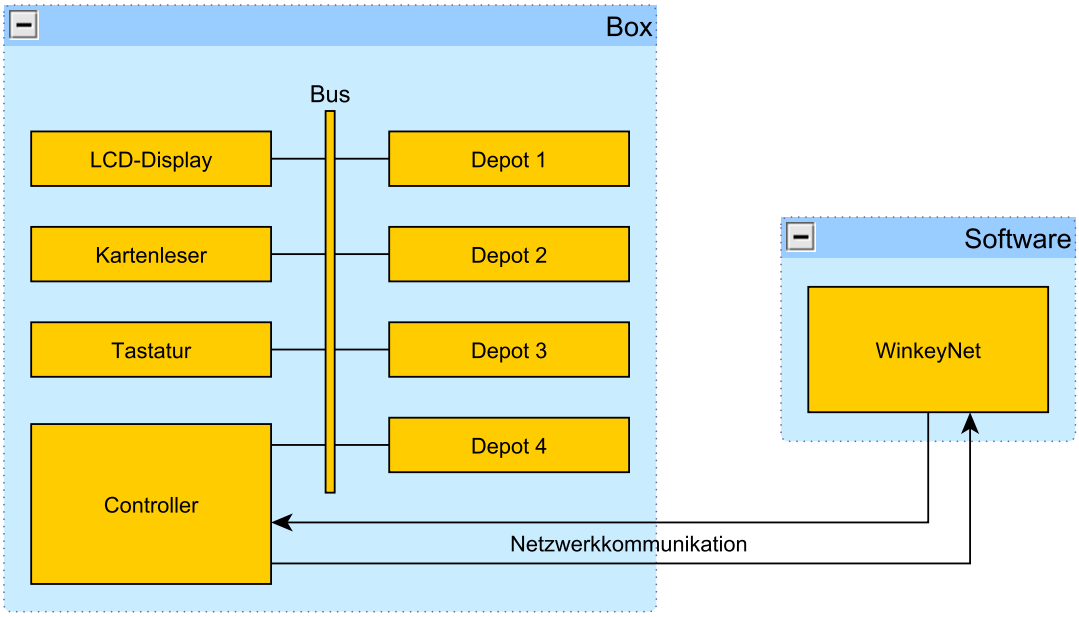


Abbildung 3.1: Alte Boxarchitektur

3.1.2 Aktuelle Boxarchitektur

Von den Kunden wurde jedoch mit aufkommender Verbreitung von Touchscreens eine intuitivere Darstellung und Bedienung gewünscht. Die Boxen sollten zudem an die Firmen-CI anpassbar sein.
Diesem Wunsch nur mit dem C165 nachzukommen war nicht möglich, da die Programmierung des Chips in Assembler erfolgte und die Implementierung einer GUI und die

Unterstützung von komplexen Benutzerinteraktionen einen gewaltigen Programmieraufwand bedeutet hätten. Auch wäre die Leistungsfähigkeit und optische Qualität der GUI durch den beschränkten Speicher des Chips miserabel ausgefallen.
Aus diesen Gründen wurde ein SBC, ein vollwertiger Computer, konzentriert auf einer einzigen Platine, in die Architektur aufgenommen. Er dient als Verbindungsschicht zwischen Hardwarecontroller und Software und stellt eine touchscreenkompatible GUI zur Verfügung.

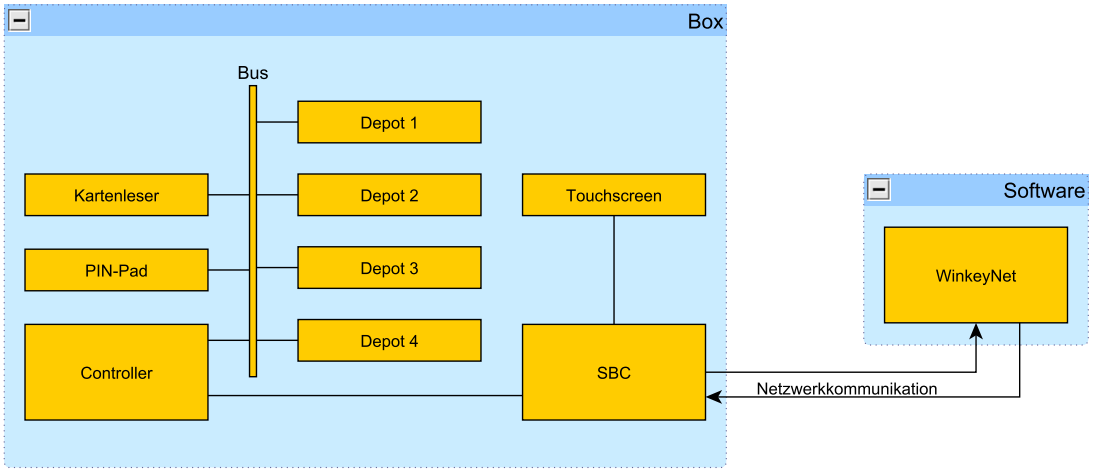


Abbildung 3.2: Neue Boxarchitektur

Durch den SBC ist es nun möglich, Hardwaretastaturen durch eine virtuelle Tastatur auf dem Touchscreen abzulösen. Auch sind benutzerfreundliche und intuitiv bedienbare GUIs darstellbar. Da der SBC zwischen Software wie z.B. WinkeyNet und dem C165 vermittelt, ist zudem eine Befehlsvirtualisierung möglich, d.h. Befehle, die nicht direkt vom Hardwarecontroller unterstützt werden, können in unterstützte Teilbefehle übersetzt werden. Auch ist nun eine Speicherung von Berechtigungen auf dem SBC möglich, z.B. mit welchem RFID-Transponder welches Depot geöffnet werden darf. Dadurch können Fächer auch während eines Verbindungsabbruchs mit WinkeyNet, das normalerweise die Berechtigungen speichert, von den authorisierten Personen geöffnet werden.

3.1.3 Fazit: Ansatzpunkt für die Virtualisierung

Es gibt zwei Teilziele bei der prototypischen Entwicklung:

1. Der Entwicklungsaufwand soll so gering wie möglich gehalten werden.
2. Mithilfe des Prototyps sollen erste Tests neuer Software durchgeführt werden können, die normalerweise eine physische Box benötigen.

Daraus ergibt sich, dass am besten mit der einfachsten Variante der Boxvirtualisierung begonnen werden sollte, denn die Virtualisierung eines SBC, der in der aktuellen Boxarchitektur hinzugekommen ist, würde durch die Virtualisierung von Benutzerrechten, sowie der Wahrung einer Trennung zwischen C165 und SBC und der Sichtbarmachung des (virtuellen) Datenverkehrs zwischen diesen, einen erheblichen Mehraufwand bedeuten. Durch die stark gestiegene Komplexitität würde die prototypische Entwicklung zudem wesentlich fehleranfälliger. [Vgl. HHI07, S. 142]
Die Virtualisierungssoftware wird deshalb die ältere Boxarchitektur ohne SBC nachbilden. Es wird die „Box“ aus Abbildung 3.1 virtualisiert, wobei das Hauptaugenmerk auf das korrekte Senden, Empfangen und Verarbeiten von Protokollen zwischen Box und Software gelegt wird.

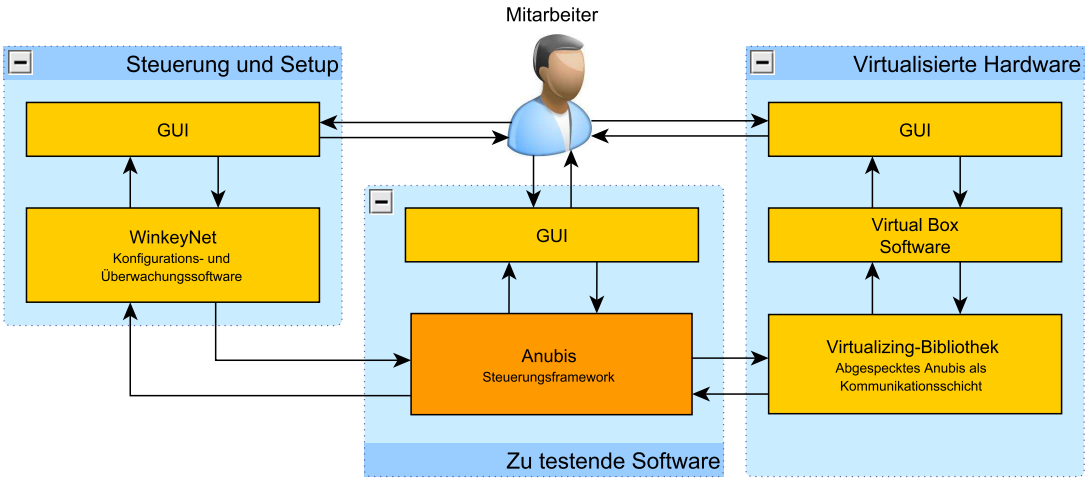


Abbildung 3.3: Architektur der Virtualisierung

In Abbildung 3.3 ist zu sehen, wie das Zusammenspiel zwischen Virtualisierungssoftware, getesteter Software (tieforange hervorgehoben) und Konfigurationssoftware funktioniert. Der Benutzer bedient dabei die GUIs von:

1. WinkeyNet, um die Box einzurichten und die Funktionsfähigkeit zu überprüfen.
2. der Virtual Box, um die Depotzustände zu überwachen, Depots zu schließen, Transponder einzulegen etc.
3. Anubis, um neue Softwarefunktionen zu testen und Aktionen auszulösen.

Anubis ist ein bereits weit entwickeltes firmeninternes Framework zur ein- und ausgehenden Kommunikation mit Boxen und zur Darstellung einer touchscreen-freundlichen GUI. Die zu entwickelnde Virtualisierungssoftware wird eine auf Anubis basierende Kommunikationsschnittstelle nutzen (genannt Virtualizing-Bibliothek) um den Entwicklungsaufwand zu verringern. Es ist jedoch abzusehen, dass das Framework um für die Virtualisierung nötige Funktionen erweitert werden muss.
Ein typischer Ablauf eines Tests könnte z.B. so aussehen: Ein Mitarbeiter erstellt in

WinkeyNet ein neues Gerät Namens „Virtual1“ mit der Box-ID 1 und konfiguriert es nach seinen Wünschen (z.B. Depotanzahl und -anordnung). Er stellt zudem ein, dass nur ein Mitarbeiter mit der Kartennummer 123 die Box bedienen darf.
Er weist die Box an, Daten an seine zu testende Software zu senden (Anubis). Anubis wiederum hat keine physische Box zur Verfügung und wird deshalb so konfiguriert, dass es Daten an die Virtualisierungssoftware sendet (und auch wieder von dort empfängt), welche den Hardwarecontroller virtualisiert. Nun startet er die Virtualisierungssoftware und konfiguriert sie so, dass die Depotanzahl und das -layout mit der Einstellung in WinkeyNet übereinstimmen. Ihm wird dann das virtuelle Gerät grafisch angezeigt, mit Statusinformationen wie eingelegten Transpondern, ob Depots geöffnet sind etc.
Er wechselt nun zu Anubis, das gerade eine Schlüsselausgabesoftware geladen hat und gibt eine falsche Kartennummer ein (die wird normalerweise über einen Kartenleser eingelesen, doch kann auch in einem Debugmodus manuell eingegeben werden). Es erscheint eine Meldung, dass die Karte nicht bekannt sei. Er gibt nun die Kartennummer 123 ein, woraufhin er in den Benutzerbereich kommt. Er weist die Software nun an, Depot 1 für eine Erstbefüllung zu öffnen. Das Depot wird daraufhin in der Virtualisierungssoftware als offen angezeigt. Er kann nun dort virtuell einen Transponder einlegen.
Währenddessen wartet Anubis auf das Schließen des immer noch geöffneten Depots. Der Benutzer schließt nach dem Einlegen des Transponders das Depot, Anubis erhält von der Virtualisierungssoftware ein Protokoll, dass das Schließen erfolgt ist. Das Depot 1 wurde nun erfolgreich geöffnet, befüllt und geschlossen mithilfe der Zusammenarbeit von WinkeyNet, der Virtualisierungssoftware und Anubis.

3.2 PHP

PHP ist eine Skriptsprache zur Erstellung von HTML-Inhalten. Diese müssen sich nicht auf HTML-Code beschränken. Praktisch jedes Dateiformat lässt sich mittels eines PHP-Skripts generieren, z.B. Bilder, Flashfilme oder PDFs. Dank der Unterstützung vieler mehr oder weniger bekannter Datenbanksysteme ist es zudem mit PHP sehr einfach, dynamische Inhalte aufgrund von Daten aus Datenbanken zu generieren.
PHP läuft auf praktisch allen häufig genutzen Betriebssystemen, darunter auch Windows, Linux, Unix und Solaris. [Vgl. Ler+07, 1 f.]
Anders als z.B. Java erzwingt PHP keine strikte Trennung von prozeduralem und objektorientiertem Code. Es ist nicht nur möglich, sondern teilweise auch äußerst hilfreich, beide Arten der Programmierung zu vermischen, um schneller zu einem gewünschten Ergebnis zu kommen. [Vgl. KÖ10, 112 f.] Denn auch wenn es möglich ist, den Programmcode rein objektorientiert zu schreiben, so würde die damit erreichbare Verbesserung der Erweiterbarkeit und Portabilität nicht immer sinnvoll sein, dafür aber ein hohes Maß an Planungs- und Programmieraufwand bedeuten. PHP geht ebenfalls darauf ein, indem einige Kernfunktionen gleichermaßen objektorientiert oder prozedural verwendbar sind (z.B. mysqli [Vgl. KÖ10, S. 418], für MySQL gibt es zudem auch PDO, ein fertiges Datenbankframework).

3.2.1 Gegenüberstellung: Prozedurale vs. Objektorientierte Programmierung

Um die Unterschiede zwischen prozeduraler und objektorientierter Programmierung aufzuzeigen, bietet sich ein Beispiel an.
Um sich zu einer Datenbank zu verbinden und eine SQL-Query abzusenden, sowie deren Ergebnis auszugeben und eine einfache Fehlerauswertung vorzunehmen, ist prozedural folgender Code notwendig:

```php
<?php
  function my_query($query)
  {
    $error = "";
    $conn = mysql_connect("localhost", "Benutzername", "Passwort");
    if (!$conn)
      $error = "Keine Verbindung zur Datenbank.";
    else {
      if (!mysql_select_db("Testdatenbank", $conn))
        $error = "Die Datenbank ist nicht vorhanden.";
      else {
        $result = mysql_query($query);
        return $result;
      }
    }

    if ($error) {
      echo "Es ist ein Fehler aufgetreten: " . $error;
      return false;
    }
  }

  $result = my_query("SELECT name FROM testtabelle WHERE id = 2");
  if ($result) {
    $name = mysql_fetch_row($result);
    if ($name)
      echo "Der Name ist: " . $name[0] . ".";
    else
      echo "Der Name ist nicht bekannt.";
  }
?>
```

Listing 3.1: Datenbankabfrage prozedural

Der objektorientierte Programmcode unter Verwendung von PDO sieht dann so aus:

```php
<?php
  class Database
  {
    private $benutzername;
    private $passwort;

    public function __construct($benutzername = "Benutzername", $passwort = "
        Passwort")
    {
      $this->benutzername = $benutzername;
      $this->passwort = $passwort;
    }

    public function my_query($query)
```

```
    {
15    try {
        $conn = new PDO("mysql:host=localhost;dbname=Testdatenbank", $this->
            benutzername, $this->passwort);
17      return $conn->query($query);
      } catch (PDOException $e) {
19      echo "Es ist ein Fehler aufgetreten: " . $e->getMessage();
        return false;
21    }
    }
23  }
  //... Code in einer anderen Klasse
25  $db = new Database; //Standardkonstruktor
  $result = $db->my_query("SELECT name FROM testtabelle WHERE id = 2");
27  if ($result) {
    $name = $result->fetchColumn();
29    if ($name)
      echo "Der Name ist: " . $name . ".";
31    else
      echo "Der Name ist nicht bekannt.";
33  }
?>
```

Listing 3.2: Datenbankabfrage mit PDO, objektorientiert

Beide Codebeispiele erzeugen die gleiche Ausgabe – nur im Fehlerfall ist die Fehlerbeschreibung in der objektorientierten Variante durch den Bezug der Fehlernachricht aus dem Objekt PDOException detaillierter. Auch können hier Fehler behandelt werden, denen in der prozeduralen Version keine Rechnung getragen wurde, da nicht nur „Keine Verbindung zur Datenbank.“ und „Die Datenbank ist nicht vorhanden.“ abgefangen werden können, sondern *alle* Fehler, die bei der Herstellung einer Datenbankverbindung und dem Absetzen und Auswerten einer SQL-Query auftreten können. Es müssen keine Prüfungsmethoden und Fehlernachrichten für alle möglichen Fehlertypen manuell programmiert werden. Dies ist ein genereller Vorteil bei der Verwendung von OOP mit einer Try-Catch-Umgebung (verwendet in Listing 3.2, Zeile 15 - 21). [Vgl. TW09, S. 231]
Als „Nebeneffekt“ (eigentlich ist dies der hauptsächliche Grund) der Verwendung von PDO funktioniert das Codestück aus Listing 3.2 auch mit anderen Datenbanksystemen. Es muss lediglich der String zum Aufbau der Verbindung ausgetauscht werden. Dazu folgt in Abschnitt 5.5 mehr.

3.2.1.1 Auswertung: Prozedurale vs. Objektorientierte Programmierung

Ohne Kenntnisse über prozedurale Programmierung ist auch keine objektorientierte Programmierung möglich, denn diese teilen sich das Ziel, Code wiederverwendbar zu machen. Genaugenommen ist die prozedurale Programmierung, wie sie die meisten PHP-Programmierer anwenden, aber gar keine prozedurale, sondern eine *funktionale Programmierung*, denn Prozeduren haben keinen Rückgabewert – im Gegensatz zu Funktionen. Da die Prozedur jedoch der Urbaustein jeglicher Wiederverwendbarkeit von Code war, hat sich diese Bezeichnung bis heute gehalten. [Vgl. Kan07, S. 34]

Durch prozedurale Programmierung wird unübersichtlicher Spaghetticode in kleine Funktionen aufgeteilt, welche an verschiedenen Stellen im Code wiederverwendet werden können. In Listing 3.1 ist es die Funktion *my_query()*, die als Parameter eine SQL-Query als String entgegennimmt und als Rückgabewert im Fehlerfall *false* zurückliefert oder im Erfolgsfall das Resultset von *mysql_query()*. Alle möglichen Fehler müssen durch if-Anweisungen wie in Zeile 6 und 9 manuell abgefangen werden. So würde z.B. ein fataler Fehler ausgelöst, wenn die Funktion ohne Parameter aufgerufen oder eine falsche SQL-Query übergeben wird.
Bei der objektorientierten Variante werden Objekte eingesetzt, um den Code besser wiederverwendbar zu machen. So kann die Klasse Database beliebig oft in beliebigen anderen Klassen instanziert werden und ist dann als Objekt mit seinen Objektmethoden verfügbar. Da das Objekt „Database“ nun auch direkt seine Methode *my_query()* mitliefert, ist der Programmcode wesentlich strukturierter.
Die Objektorientierung bringt zudem den Vorteil, dass sich die Sichtbarkeit von Methoden und Daten verändern lässt. [Vgl. Kan07, S. 200] Dadurch ist es in Listing 3.2 z.B. nicht möglich, die Eigenschaften $benutzername und $passwort außerhalb des Konstruktors zu beeinflussen. Diese Sichtbarkeitssteuerung bietet vor allem Vorteile, wenn mehrere Klassen ein Objekt gleichzeitig benutzen. Denn durch das „Verbot“, bestimmte Eigenschaften einfach zu ändern, lässt sich reglementieren, wer wann etwas ändern darf. So ließen sich z.B. Get- und Set-Methoden für $benutzername und $passwort entwickeln, die darauf achten, dass gerade keine Datenbankverbindung aktiv genutzt wird.

3.2.1.2 Fazit

Es gibt keinen Grund, sich gegen objektorientierte Programmierung zu entscheiden. OOP erhöht die Übersichtlichkeit und Wiederverwendbarkeit des Programmcodes und erfordert dafür einen (meist) geringen Mehraufwand, so z.B. die Kapselung der Funktionen in Klassen. [Vgl. Kan07, S. 229] Die Klasseneigenschaften sollten zudem so gewählt werden, dass diese die Klasse gut beschreiben. Denn auch treffende Klasseneigenschaften verbessern die Nutz- und Änderbarkeit. Was läge näher, als in Listing 3.2 als nächstes Get- und Set-Methoden für $benutzername und $passwort zu entwerfen? Oder sollen diese Eigenschaften außerhalb des Konstruktors niemals verändert werden dürfen? Die Klasse ließe sich zudem problemlos um weitere Methoden erweitern, z.B. um Vereinfachungen für SQL-Befehle wie *my_select()*, *my_update()* u.a.
Sofern die Objektorientierung eingehalten wird, kann ein Programm aus vielen Objekten zusammengestellt werden, die miteinander interagieren können. Dies fördert die Änderbarkeit und Übersichtlichkeit.

3.2.2 Trennung von Darstellung und Programmlogik

Das bekannteste Modell zur Trennung von Darstellung und Programmlogik ist das Model-View-Controller-Modell. Dieses Modell beinhaltet laut Kannengiesser [Kan07, S. 419]:

- das *Model*, welches alle darzustellenden Daten enthält.
- den *View*, welcher die Daten aus dem Model abruft und darstellt.
- den *Controller*, der auf Benutzeraktionen reagiert und den View-Ablauf steuert.

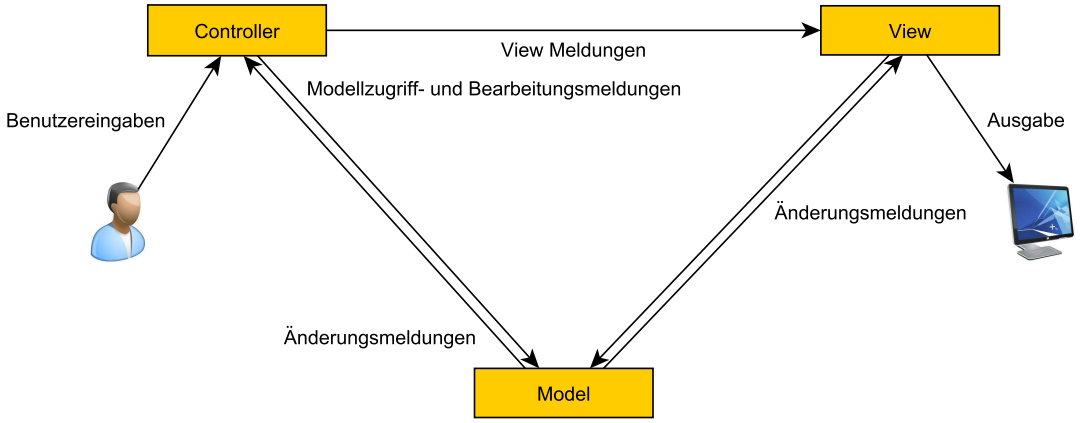


Abbildung 3.4: MVC-Modell [Kan07, S. 419]

In PHP kann dieses Schema gut umgesetzt werden, jedoch wird „[...] bei der Implementierung von Dialogen und Fenstern häufig die Steuerung und die Präsentation zusammengefasst.“ [Kan07, S. 419] Das ist gängige Praxis, da sich der Arbeitsaufwand für eine Trennung bei Dialogen kaum lohnt. Denn die Darstellung des Dialogs und die Interaktion mit dem Benutzer und die daraus resultierenden Aufrufe des Controllers sind eng verzahnt, eine Aufsplittung kaum sinnvoll. Hingegen ist die Trennung des Models sehr einfach: es ist durch die Datenbankanbindung an PHP bereits gegeben.
Die zwei Hauptmethoden in PHP, um nun Präsentations- und Steuerungscode nicht unnötig zu vermischen, sind:

1. die Nutzung von Templates.
2. eine saubere Programmierweise, in der Control und View nacheinander in einer Datei abgearbeitet werden.

3.2.2.1 Verwendung eines Templates

Werden Templates verwendet, so lesen Skripte die Daten aus der Datenbank ein und verarbeiten diese. Um nun diese Daten darzustellen und dem Benutzer Interaktionsmöglichkeiten zu geben, wird ein HTML-Grundgerüst mit Platzhaltern verwendet, die durch

generierten HTML-Code ersetzt werden.
Ein einfaches Template könnte z.B. so aussehen:

```
<html>
<head>%%HEAD%%</head>
<body>%%BODY%%</body>
</html>
```

Listing 3.3: Template, template.tpl

%%HEAD%% und %%BODY%% sind Platzhalter für Daten, die an diese Stelle eingesetzt werden sollen. Um nun Daten einzusetzen, wäre dieses kurze PHP-Stück ausreichend:

```
<?php
  $template = file_get_contents("template.tpl");
  $template = str_replace("%%HEAD%%", "<title>Testtitel</title>", $template);
  $body = "<table><tr>";
  for($i = 0; $i < 10; i++)
    $body .= "<td>" . $i . "</td>";
  $body .= "</tr></table>";
  $template = str_replace("%%BODY%%", $body, $template);
?>
```

Listing 3.4: Daten in Template einsetzen

Dies würde eine Webseite mit dem Titel „Testtitel“ und einer Tabelle mit 10 Spalten, die jeweils eine Zahl aufsteigend von null bis neun enthalten, erzeugen. Dies ist natürlich ein sehr einfaches Beispiel, doch statt der einfachen Vorgaben und der Schleife könnte an dieser Stelle auch eine Datenbankabfrage stehen, was einen Zugriff auf das Model aus dem Controller bedeuten würde. Der View ist das Template, welches mithilfe von *file_get_contents()* als String in $template gespeichert wurde. Die Daten aus dem „Model“ werden anschließend mit *str_replace()* eingesetzt.

3.2.2.2 Control und View nacheinander

Vor allem bei Dialogen ist diese Schreibweise sehr beliebt und obwohl Control und View innerhalb derselben Datei vorkommen, leidet die Übersichtlichkeit nicht. Es ist lediglich etwas Programmcode nötig, der bei jeder Ausführung die Daten im View verändert und überprüft, ob überhaupt Daten übergeben worden sind und ob diese korrekt sind. Im einfachsten Fall, bei einem Formular mit nur einem Feld, könnte das so aussehen:

```
<html>
<head></head>
<body>
  <?php
    if((int) $_POST['alter'] > 0)
      my_query("INSERT INTO alter VALUES (" . (int) $_POST['alter']. ")");
  ?>
  <form method="POST" name="alterseingabe">
    Alter: <input type="text" name="alter"><br />
    <input type="submit">
  </form>
</body>
</html>
```

Listing 3.5: Control und View nacheinander

Der View, also der HTML-Code, ist nun schon verzahnter mit dem dynamischen PHP-Code, doch die Trennung ist noch gut ersichtlich und die Art der Zusammenarbeit weiterhin logisch. Nach einer Eingabe im View und anschließendem Absenden wird das Alter in der Datenbank hinterlegt. Der View übernimmt hier im ersten Schritt auch die Arbeit des Controllers. Das Model ist die Datenbank, welche mit der *my_query()*-Funktion aus Listing 3.1 angesprochen wurde.
An diesem Beispiel wird gut ersichtlich, dass sich eine weitere Trennung kaum lohnen würde. Die Einheit aus Dialog (= View) und Controller würde verlorengehen und an Übersichtlichkeit verlieren. Der Programmcode würde sich auf zwei Dateien aufteilen, wobei jede weitere Datei die Mehrarbeit des Suchens und Öffnens mit sich bringt.

3.2.2.3 Fazit

Die Trennung des Models von den restlichen Bestandteilen ist in PHP durch die Datenbankanbindung relativ einfach. Beim Trennen von Controller und View muss zudem abgewägt werden, ob sich eine absolut strikte Trennung rentiert oder ob damit eine zusammengehörige Einheit aufgespalten wird, die allein übersichtlicher gewesen wäre als die getrennte Variante. Besonders bei Interaktionen mit dem Benutzer, z.B. in Dialogen, ist die Trennung häufig wenig sinnvoll. [Vgl. Kan07, S. 419]
Durch einen guten Programmierstil, der Controller- und Viewteile möglichst nicht miteinander mischt und die Nutzung von Datenbankabstrahierungslayern wie z.B. PDO (siehe Abschnitt 5.5) ist eine gute Codeübersichtlichkeit erreichbar und die Entwicklungszeit des Prototypen wird nicht unnötig verlängert. Der lauffähige Prototyp steht im Vordergrund.

3.2.3 Serviceprogrammierung

Ein Service ist eine Software, die im Hintergrund läuft und mit einer Anwendung im Vordergrund kommuniziert. Diese Software wird auch „Dienst“ genannt. Ein Beispiel für

solch einen Service ist der DHCP-Client von Windows. Dieser läuft im Hintergrund und bearbeitet DHCP-Anfragen, während der Benutzer selbst (unter Windows Vista/7) im „Netzwerk- und Freigabecenter“ Einstellungen vornimmt.

3.2.3.1 Probleme bei der Serviceprogrammierung in PHP

Wie lässt sich nun also so ein Service in PHP programmieren? Die Antwort ist: unter normalen Umständen gar nicht. Drei Probleme verhindern dies normalerweise:

1. PHP ist nicht in der Lage, Windows Executables (exe-Dateien) zu erstellen, die dann direkt von Windows als Service im Hintergrund geladen werden können.
2. In PHP gibt es eine „max execution time“, eine maximale Ausführungs- und Interpretierungszeit, die verhindert, dass ein PHP-Skript unendlich lang arbeitet. [Vgl. Gil08, S. 544]
3. PHP-Skripte sind normalerweise an die GUI, die beim Client angezeigt wird, gebunden. Wird die Ausführung der GUI gestoppt oder unterbrochen (z.B. durch einen Klick auf „Abbrechen“ im Browser oder durch Schließen des Browserfensters), wird auch die Skriptabarbeitung auf dem Server abgebrochen (mit allen negativen Auswirkungen eines nicht vollständig abgearbeiteten Skripts). [Vgl. Ler+07, S. 467]

PHP ist somit eigentlich ungeeignet, um Services zu schreiben und trotzdem ist es eingeschränkt möglich und sogar sinnvoll.

3.2.3.2 Keine ausführbaren Dateien

Über Punkt 1 kann hinweggesehen werden, sofern es sich bei der zu programmierenden Anwendung um ein rein webbasiertes Projekt handelt. Das ist bei der prototypischem Implementierung der Virtualisierungssoftware der Fall. Alle Teile dieser Software sind PHP-Skripte und laufen im Browser ab. Sofern Apache und der PHP-Interpreter laufen, können PHP-Skripte im Browser geöffnet werden.
Der Service könnte dann z.B. durch eine Autostart-Aktion beim Starten von Windows angestoßen werden und er könnte im Hintergrund arbeiten. Es muss aber auf Punkt 3 geachtet werden.

3.2.3.3 Maximale Ausführungszeit

Punkt 2 ist hingegen ein ernstes Problem. Denn wie soll ein PHP-Skript als Service laufen, wenn es, bei Standardeinstellungen, nach 30 Sekunden automatisch abgebrochen wird, um das System vor endlos laufenden Skripts (die endlos Ausgaben generieren und somit den Speicher füllen könnten) zu schützen? Die Lösung ist relativ einfach. Sofern der Safemode abgeschaltet ist, lässt sich mit *set_time_limit(0)* die Maximallaufzeit aufheben.

Dieses Vorgehen birgt allerdings Gefahren, denn nicht umsonst gibt es diese Sicherheitsvorkehrung.

```
1 <?php
    set_time_limit(0);
3   ignore_user_abort(true);
    while(1) {
5     echo ".";
    }
7 ?>
```

Listing 3.6: Endlosschleife ohne Schutz durch maximale Ausführungszeit

Das Skript aus Listing 3.6 würde in einer Sekunde eine enorme Menge Output produzieren und damit erst aufhören, wenn ihm „der Boden unter den Füßen weggezogen“ würde, nämlich der Apache-Server. Es gibt keine Möglichkeit, dieses Skript abzubrechen, denn es hat nicht wie ein gewöhnliches ausführbares Programm unter Windows einen entsprechenden Prozess, der über den Task Manager beendet werden könnte. Der Apache-Server müsste zumindest neugestartet werden, damit er dieses Skript nicht mehr ausführt.
Damit dieses Problem nicht auftritt, muss mindestens eine Ausstiegsbedingung definiert werden, damit das Skript im Problemfall beendet werden kann. Als Abbruchbedingung bietet sich z.B. der Test auf eine Datei an. Wird diese vom Skript erkannt, bricht es ab:

```
1 <?php
    set_time_limit(0);
3   ignore_user_abort(true);
    while (1) {
5     if (file_exists(dirname(__FILE__) . "/stop"))
        break; //Endlossschleife verlassen
7     echo ".";
    }
9 ?>
```

Listing 3.7: Endlosschleife ohne Schutz durch maximale Ausführungszeit

Die Funktion *dirname(__FILE__)* ermittelt das Arbeitsverzeichnis der aktuell interpretierten Datei, deren kompletter Pfad inklusive Dateiname in der magischen Konstante __FILE__ steht. Es wird dann auf eine Datei namens „stop“ in dem Verzeichnis geprüft, in der das Skript gespeichert ist (Zeile 5). Existiert die Datei, wird die Schleife abgebrochen (Zeile 6).

3.2.3.4 Bindung zwischen GUI und PHP-Skript

Das Problem aus Punkt 3 lässt sich ähnlich einfach lösen wie das Problem mit der maximalen Ausführungszeit (siehe Unterunterabschnitt 3.2.3.3). Auch hierfür gibt es eine Möglichkeit, die Bindung zwischen GUI und PHP-Skript zumindest so zu lösen, dass Benutzerabbrüche (z.B. ein Klick auf „Abbrechen“ im Browserfenster oder das Schließen

des Browserfensters) ignoriert werden. Der Befehl dazu lautet *ignore_user_abort(true)* und wurde bereits in Listing 3.6 angewendet. Ohne diesen Befehl hätte das Skript ganz einfach über „Abbrechen“ oder Schließen des Browserfensters gestoppt werden können, eine Implementierung einer Abbruchbedingung, wie in Listing 3.7 geschehen, wäre nicht nötig gewesen.

3.2.3.5 Fazit

Mit ein paar Einschränkungen lassen sich auch mit PHP Services entwickeln, es ist allerdings nötig, dass dazu ein Apache-Server mit PHP-Unterstützung läuft. Es muss zudem beachtet werden, dass PHP-Skripte standardmäßig eine maximale Laufzeit haben, nach der sie abgebrochen werden. Des Weiteren muss sichergestellt werden, dass das PHP-Skript, wenn es als Service laufen soll, nicht abgebrochen wird, wenn der Benutzer das Browserfenster, in dem er das PHP-Skript gestartet hat, schließt oder auf „Abbrechen“ klickt. Außerdem muss es eine implementierte Abbruchbedingung geben.

3.3 Java als Alternative zu PHP

Java ist eine besondere Programmiersprache, denn Programmcode, der in Java geschrieben wurde, wird weder interpretiert, noch so kompiliert, dass er einfach in Windows oder einem anderen Betriebssystem ausführbar ist. Java-Code wird vom Java-Compiler in Bytecode übersetzt. Dieser Bytecode ist vom Betriebssystem unabhängig. Das einzige, das für seine Ausführung benötigt wird, ist eine virtuelle Maschine für das entsprechende Betriebssystem. Im Falle von Java ist dies die JVM, welche es für eine große Anzahl von Betriebssystemen gibt. [Vgl. Fla05, S. 3]
Ist diese Programmiersprache besser für die prototypische Entwicklung einer Virtualisierungssoftware geeignet?

3.3.1 Objektorientierte Programmierung

Im Gegensatz zu PHP erzwingt Java praktisch die objektorientierte Programmierung. Dies begründet sich darin, dass Java keinen ungekapselten Programmcode unterstützt. Sämtlicher Quellcode muss in Klassen und darin wiederum in Methoden verkapselt sein. Auch müssen die Methoden und Variablen einen Sichtbarkeitsmodifikator (private, static, protected) besitzen. Selbst wenn kein solcher Modifikator vom Programmierer gesetzt wird, haben die Methoden oder Variablen *package*- bzw. *default*-Sichtbarkeit (ein Mittelweg zwischen private und public). [Vgl. Fla05, S. 146] Das, was der prozeduralen Programmierung in PHP am nächsten kommt, ist eine schwache Objektorientierung, wie in Listing 3.9 gezeigt. Hier wird zwar eine Klasse verwendet und auch eine Klassenmethode, diese sogenannte „Main-Methode“ wird aber beim Aufruf des Java-Programms automatisch aufgerufen. Dies macht eine prozedural-ähnliche Programmierung möglich.

Die Grenze zur Objektorientierung ist allerdings dünn und schnell überschritten. Denn schon beim Definieren neuer Methoden werden Klassenmethoden mit eingeschränkter Sichtbarkeit geschaffen und auch wenn der Konstruktor der Klasse nicht definiert wird, legt Java einen (leeren) Standardkonstruktor an, der auch aufgerufen werden kann. [Vgl. Fla05, 109 f.]
Es ist somit ein streitbares Thema, ob in Java überhaupt nicht-objektorientiert programmiert werden kann und ist von der Definition des Programmierers für „Objektorientierung“ abhängig.

Durch die erzwungene Kapselung ist kein Programmcode wie dieser möglich, mit dem einfachen Ziel „Hallo Welt“ auf dem Bildschirm auszugeben:

```
1 System.out.println("Hallo Welt!");
```

Listing 3.8: „Hallo Welt!“ in Java, nicht ausführbar

Ein korrektes Java-Programm, welches „Hallo Welt!“ ausgibt, müsste mit höherem Aufwand geschrieben werden. Dieser Code kann noch als prozedural betrachtet werden, da keine Klassen instanziert oder Methoden mit eingeschränkter Sichtbarkeit definiert werden:

```
1 class HalloWelt
  {
3   public static void main() {
      System.out.println("Hallo Welt!");
5   }
  }
```

Listing 3.9: „Hallo Welt!“ in Java, korrekt

In PHP hätte es (prozedural) lediglich dieser Zeilen bedurft:

```
<?php
2   echo "Hallo Welt!";
?>
```

Listing 3.10: „Hallo Welt!“ in PHP

Auch wenn in Listing 3.9 kein Konstruktor zu sehen ist, so legt Java im Hintergrund den parameterlosen Standardkonstruktor HalloWelt() an. Die Grenze zur Objektorientierung ist somit praktisch schon überschritten. Soll der Programmcode nun eine ordentlich objektorientierte Form erhalten, so lautet er:

```
1  class HalloWelt
   {
3    public HalloWelt() {}

5    private void begruessung() {
       System.out.println("Hallo Welt!");
7    }

9    public static void main() {
       HalloWelt halloWelt = new HalloWelt();
11     halloWelt.begruessung();
     }
13 }
```

Listing 3.11: „Hallo Welt!“ in Java, korrekt und objektorientiert

Beim Start dieses Programms würde die Main-Methode automatisch durch die JVM aufgerufen. Anschließend erzeugt die Main-Methode eine Instanz der HalloWelt-Klasse. Diese Instanz ist nun das HalloWelt-Objekt und eine Referenz darauf wird in der Variable halloWelt gespeichert. Über diese Referenz lassen sich nun die Objektmethoden aufrufen (also die Methoden der instanzierten Klasse). Durch den Aufruf von `halloWelt.begruessung()` wird die *private* deklarierte und somit nur innerhalb dieser Klasse sichtbare Methode `begruessung()` aufgerufen, welche „Hallo Welt!“ auf dem Bildschirm ausgibt.

3.3.2 Trennung von Darstellung und Programmlogik

Auch in Java kommt hauptsächlich das MVC-Modell aus Abbildung 3.4 zum Einsatz. Es liegt, wie auch bei PHP und anderen Programmiersprachen mit Objektorientierung, lediglich im Ermessen des Programmierers, wie stark er die einzelnen Teile voneinander trennt und wie lohnenswert die Trennung ist.
Um die Datenbank zu abstrahieren, bietet auch Java eine hervorragende Datenbankintegration mittels JDBC, einer Schnittstelle zwischen der SQL-Datenbank und dem Programm. [Vgl. KS09, S. 988] Weitere Abstrahierungen, z.B. Methoden des Models um in bestimmte Tabellen zu schreiben oder andere CRUD-Befehle auszuführen, sind problemlos möglich.
Die Parallelen zu PHP sind auch bei Controller und View zu erkennen: so ist es auch in Java üblich diese in Dialogen nicht zu trennen, da der Code des Controllers direkt auf die Benutzereingaben reagiert und den View-Ablauf steuert. Die heutigen IDEs bringen GUI-Designprogramme mit, die einem das „Zusammenklicken“ einer GUI ermöglichen.

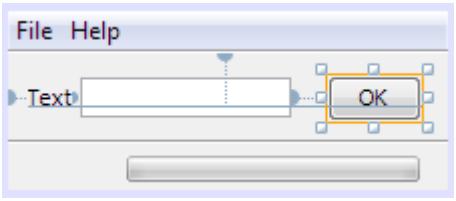


Abbildung 3.5: In NetBeans 6.9 designte GUI

Den einzelnen Elementen wie Buttons, Textfeldern etc. lassen sich dann Aktionen zuweisen, z.B. bei einem Klick oder einer Texteingabe. Der Codeblock der daraufhin automatisch generiert wird, wird direkt in der Quellcodedatei gespeichert, in der auch der generierte Programmcode für den View/die GUI gespeichert ist. Um dem Programmierer die Übersicht zu erleichtern, werden die Quellcodeblöcke für das GUI-Design deshalb standardmäßig ausgeblendet.

```
100 ⊞    Generated Code
221
222 ⊟    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
223          jTextField1.setText("Hallo Welt");
224      }
```

Abbildung 3.6: Von NetBeans 6.9 generierter GUI-Code und Code für den Button

Dies entspricht der sauber getrennten Programmierung von Controller und View bei PHP. Es sind auch die Zeilennummerierungen in Abbildung 3.6 zu beachten. Etwa 120 Zeilen Code für die GUI wurden von NetBeans generiert und „versteckt“ (mit einem Klick auf das „+“ kann der generierte Quellcode angezeigt werden).

3.3.3 Serviceprogrammierung

Im Gegensatz zu PHP gibt es bei Java-Programmen kaum Einschränkungen bei der Programmierung von Services. Javaprogramme sind, vorausgesetzt Java (und somit das JRE, das Java Runtime Environment) ist installiert, über einen Doppelklick unter Windows ausführbar und laufen als eigenständige Prozesse, sind also nicht wie PHP-Skripte an einen Webserver gebunden.
Für Java existieren zudem auch Frameworks um Java-Programme in echte Windows-Dienste umzuwandeln, welche dann beim Start von Windows nicht als Prozess, sondern als Dienst ausgeführt werden. [Vgl. Roe09] Diese Dienste haben den Vorteil, dass sie vollkommen automatisch und ohne sichtbare GUI im Hintergrund laufen. Diese Dienste werden über Systemsteuerung → Verwaltung → Dienste kontrolliert. Das Starten, Stoppen oder Neustarten ist von dort jederzeit möglich, allerdings können sie nicht über eine GUI näher konfiguriert werden. Dafür müsste ein weiteres Programm mit GUI geschrieben werden, das die Einstellungen des Dienstes ändern kann.
Da das Virtualisierungsprojekt mit möglichst geringem Aufwand programmiert werden soll, kann auf die Umwandlung des Javaprogramms in einen Dienst auch verzichtet werden. Dieses lässt sich dann beispielsweise über die Autostartfunktion von Windows starten. Der Vorteil dieser Methode ist, dass das Programm selbst eine GUI haben darf, es wird kein weiteres Programm zur Einrichtung benötigt. Der Nachteil dieser Methode ist, dass das Programm wie jeder andere Prozess vom Benutzer oder anderen Programmen geschlossen werden kann. Die Ausfallsicherheit nimmt somit ab. Doch vorausgesetzt, dass das Programm auf einem Serverrechner läuft, der nicht zum normalen Arbeiten verwendet wird, ist dieses Risiko vernachlässigbar.

3.3.4 Fazit

Java verlangt vom Programmierer eine wesentlich objektorientiertere Vorgehensweise. Sämtlicher Code muss in Klassen und Methoden gekapselt werden. Auch sind unter Java Anpassungen für einen Windows-Service notwendig. Für einen echten Service müsste das Javaprogramm mit Hilfe eines anderen Programms namens JSL gekapselt werden, um eine von Windows als Dienst ladbare Exe-Datei zu erhalten. Läuft das Javaprogramm als normaler Prozess und nicht als Dienst, so ist mit einer leicht gesteigerten Ausfallwahrscheinlichkeit zu rechnen, die im Serverbetrieb aber vernachlässigt werden kann. Sie ist in etwa mit der eines im Hintergrund laufenden PHP-Skripts, das von einem Webserver abhängig ist, vergleichbar. Bei Ausfall des Webservers bzw. des JRE sind beide Programme nicht mehr nutzbar, bis der Ausfall behoben ist.
Eine saubere Programmierung in Anlehnung an das MVC-Modell aus Unterabschnitt 3.2.2 ist ebenfalls in beiden Programmiersprachen möglich, wobei PHP durch seine zwanglosere Syntax zu einem Bruch mit diesem Modell verführt.
In Anbetracht dessen, dass die in vorigen Abschnitten angesprochene Anubis-Bibliothek bereits in PHP vorliegt und der Fokus auf größtmöglicher Entwicklungsgeschwindigkeit und -effizienz liegt, ist eine Programmierung in PHP von Vorteil. Die prozedural und objektorientiert gemischte Programmierung räumt Freiheiten bei der Programmgestaltung ein und ermöglicht schnelle erste Erfolge. Durch das ebenfalls PHP-orientierte Entwicklerteam in der Firma können zudem Vorschläge der Mitarbeiter besser umgesetzt werden.
Die Virtualisierungssoftware wird somit in PHP implementiert werden.

3.4 Datenübertragung übers Netzwerk

Wie in Abbildung 3.3 zu sehen, ist viel Datenverkehr über das Netzwerk zu erwarten. Einzelne Teile des Testaufbaus, also WinkeyNet, Anubis und die Virtualisierungssoftware, können auf unterschiedlichen PCs eingesetzt werden. Um die Daten auszutauschen, wird auf Ethernet-Netzwerkverbindungen gesetzt.

3.4.1 Warum Ethernet-Netzwerkverbindungen?

Ethernet ist die am weitesten verbreitete Netzwerktechnologie. Schätzungsweise sind „mehr als 90 Prozent aller Netzwerkanschlüsse Ethernet-Verbindungen“. [Lar05, S. 206] So kommt es, dass auch in diesem Betrieb Fast- und Gigabit-Ethernet-Verbindungen zur Kommunikation zwischen Boxen und PCs untereinander verwendet werden. Die Vorteile von Ethernet sind dabei klar:

- Die Kabel und Netzwerkkomponenten sind sehr günstig, da hoher Konkurrenzdruck bei den Herstellern für regen Wettbewerb sorgt. [Vgl. Spu00, S. XII]

- Das Netzwerk ist gut skalierbar. Ab einem bereits relativ niedrigen Kabelstandard (Cat5e) kann die Datenübertragungsgeschwindigkeit durch Austausch der Netzwerkkomponenten von 100 auf 1000 MBit/s angehoben werden. Zudem sind Ethernetnetzwerkkarten abwärtskompatibel. Ein Computer mit einer Fast-Ethernet-Netzwerkkarte (100 MBit/s) kann problemlos mit einem Switch mit Gigabit Ethernet (1000 MBit/s) verbunden werden (und sendet und empfängt dann mit 100 MBit/s). [Vgl. Spu00, XII f.]
- Das Netzwerk ist sehr zuverlässig. Im Fehlerfall kommen nur wenige Kabel und Netzwerkkomponenten in Frage, die den Fehler auslösen können. Im einfachsten Fall wird das Kabel oder das Gerät ausgetauscht um die Funktionsfähigkeit wiederherzustellen. Da alle Komponenten standardisiert sind, können auch problemlos Komponenten anderer Hersteller eingesetzt werden. [Vgl. Spu00, S. XIII]
- Für Ethernet sind gute Managementtools vorhanden, die es möglich machen, ein ganzes Ethernetnetzwerk von einem Arbeitsplatz aus zu kontrollieren. Dies ist z.B. mit dem SNMP-Protokoll möglich, sofern die eingesetzten Netzwerkkomponenten es unterstützen. [Vgl. Spu00, S. XIII]

Der größte Konkurrent von Ethernet ATM scheidet bereits bei der Betrachtung des Kostenkapitels aus. So ist es günstiger ein Gigabit-Ethernet-Netzwerk mit 1000 MBit/s aufzubauen, als auf ATM-Komponenten umzusteigen. [Vgl. Inf02, S. 10-6] Auch werden die mit ATM-Komponenten teuer erkauften Vorteile wie z.B. Paketpriorisierung für bestimmte Inhalte (z.B. Videos oder VoIP nicht benötigt. [Vgl. Inf02, S. 10-4]

3.4.2 TCP vs. UDP

Aufgrund der speziellen Art und Weise in der Daten zwischen den verschiedenen Programmteilen der Virtualisierungssoftware verschickt werden, ist keine spezielle Auseinandersetzung mit den einzelnen Transportprotokollen notwendig. Genaue Erläuterungen hierzu finden Sie in Abschnitt 4.4.

4 Konzeption der Virtualisierungslösung

Durch die bereits in Abbildung 3.3 analysierte Architektur der Virtualisierungssoftware kann bei der Konzeptionierung der Software auf die bereits vorhandene Architektur der Virtualizing-Bibliothek, Anubis sowie WinkeyNet gesetzt werden. Anubis und WinkeyNet sind gegeben. Das neue Softwareprodukt muss nach erfolgter Konfiguration mit diesen zusammenarbeiten.
Die Konzeptionierung beschränkt sich deshalb auf eine Steuerungssoftware mit GUI, die mithilfe der Virtualizing-Bibliothek, die gegebenenfalls erweitert werden muss, mit Anubis und WinkeyNet kommuniziert. Das Kommunikationsprotokoll ist bekannt. Im Anhang A befindet sich eine Softwarespezifikation, die der Entwicklung der Virtualisierungssoftware vorausging. Im Nachfolgenden sollen vor allem die wichtigsten Konzept- und Designentscheidungen getroffen und erläutert werden.

4.1 Bedienoberfläche (GUI)

Die GUI soll einfach und funktional sein. Eine weitere Anforderung ist eine gute Anpassbarkeit an zukünftige Anforderungen, z.B. der einfache Austausch von Buttons und Grafiken, Anpassung der Größen, Abstände etc. von Elementen. Um dies zu erreichen, wird der von PHP generierte HTML-Code mit CSS, einer Formatierungssprache für HTML, layoutet. CSS bietet eine sehr gute Browserkompatibilität, wodurch die Virtualisierungssoftware in Internet Explorer (ab Version 6), Firefox und anderen Browsern weitgehend identisch aussieht. Nur in sehr alten Browsern kommt es zu Darstellungsfehlern, da diese die CSS-Vorgaben nicht beachteten oder falsch interpretierten. Da die Virtualisierungssoftware jedoch nur mit Firefox und dem Internet Explorer ab Version 7 eingesetzt werden soll, wird das Layout aber auch nur mit diesen Browsern kontrolliert.

4.1.1 Konfigurationsseite

Die Konfigurationsseite soll einen einfachen Zugang zu allen Einstellungen bieten, die für eine funktionsfähige virtualisierte Box nötig sind. Dies umfasst:

- IP-Einstellungen (IP-Adresse, Subnetzmaske, Standardgateway)
- Einstellungen zur Konfiguration der Virtualizing-Bibliothek (IP und Port an die Protokolle gesendet werden, Box-ID)
- Webservereinstellungen (Port des Webservers)
- Layout der Box (Boxteile der Größe $x * y$ definierbar)

Aus diesen Punkten lässt sich unter der Voraussetzung, dass die GUI mit HTML geschrieben und mit CSS layoutet sein soll, ein GUI-Entwurf ableiten. Dieser Entwurf wurde mit dem Programm Pencil[1] erstellt, einer Erweiterung für Firefox, die es möglich macht, Webseitenlayouts komfortabel zu planen. Es handelt sich nicht um einen Screenshot.

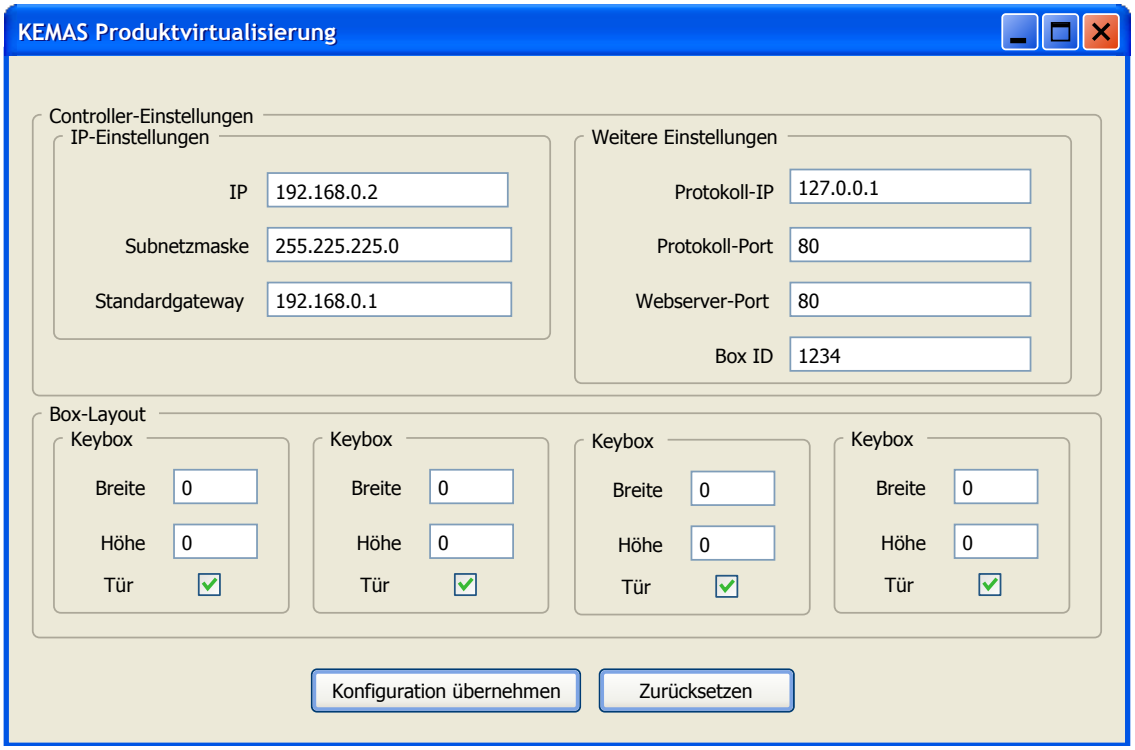


Abbildung 4.1: Entwurf der GUI der Konfigurationsseite

Die Überlegung hinter diesem Design war, dass alle wichtigen Einstellungen auf einer Seite konzentriert sind. Somit sind nur wenige Klicks nötig und der Benutzer kann bequem mit der Tabulatortaste durch die Felder wechseln.
Die beschrifteten Rahmen wie „IP-Einstellungen“, „Weitere Einstellungen“ etc. trennen die Einstellungsbereiche optisch klar voneinander ab. Diese Trennung vereinfacht das Erfassen von Zusammenhängen zwischen einzelnen Eingabefeldern. [Vgl. Hai06, S. 101]
Ein Klick auf „Zurücksetzen“ setzt alle Felder auf ihre Werte zurück, die sie vor Änderungen durch den Benutzer hatten. Ein Klick auf „Konfiguration übernehmen“ leitet den Benutzer nach einer Gültigkeitsprüfung zur Darstellungsseite weiter.

4.1.2 Darstellungsseite

Die Darstellungsseite soll in der ersten Version so einfach wie möglich gehalten werden. Die große Überschrift soll die Box-ID anzeigen, die das virtuelle Gerät hat, um die Übersicht zu erleichtern, falls mehrere virtuelle Boxen existieren. Unter der Überschrift

[1] Erhältlich unter `http://pencil.evolus.vn/en-US/Home.aspx`

sollen die einzelnen Depots als Rechtecke angezeigt werden. Der jeweilige Status (leer, korrekter Transponder (nachfolgend auch Dallas genannt), falscher Transponder) ist an einem farbigen Kreis pro Fach (der für eine LED steht) ablesbar. Die Farben sind mit gelb für leer, grün für korrekt und rot für falsch festgelegt. Das Layout des Geräts, also wie die einzelnen Depots angeordnet sind, hängt vom eingestellten Layout auf der Konfigurationsseite ab.
Weiterhin sollen Aktionen und Einstellungen für einzelne Depots sofort vorgenommen werden können. Bei einem Klick auf ein Depot soll sich deshalb eine Eigenschaftsseite öffnen, auf der der Depotstatus eingesehen werden kann und Aktionen vorgenommen werden können. Im Detail sind diese in Unterabschnitt 4.1.3 erläutert.

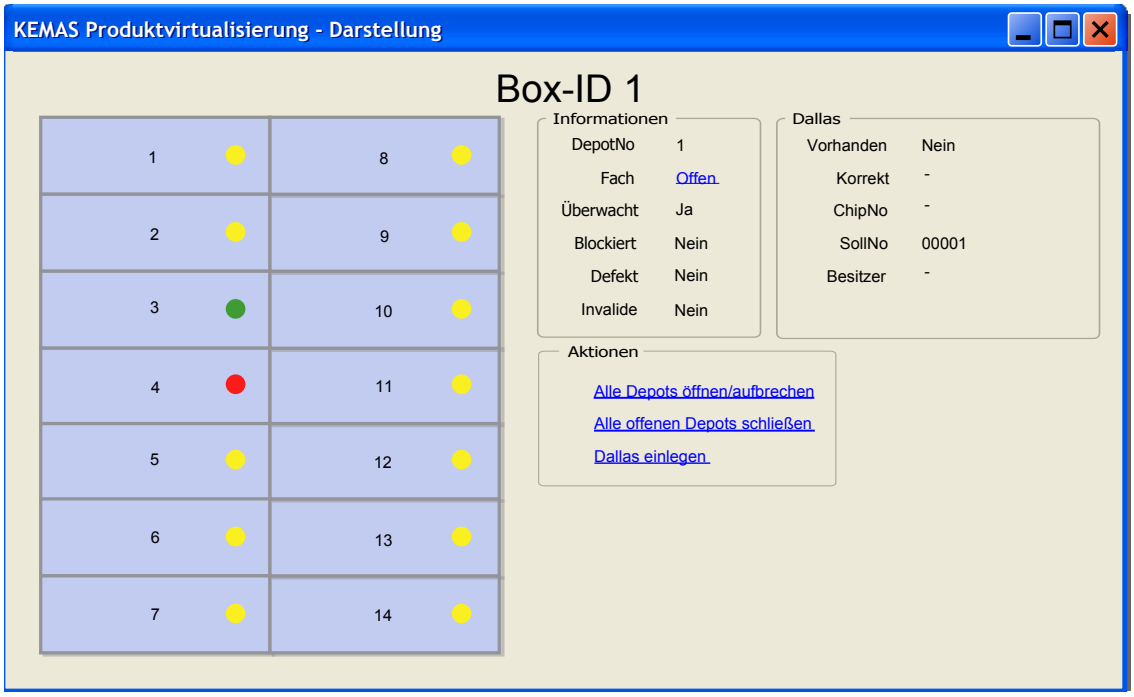


Abbildung 4.2: Entwurf der GUI der Darstellungsseite

Die Abbildung 4.2 zeigt die Darstellungsseite nach einem Klick auf Depot 1.

4.1.3 Depoteinstellungsseite

Die Depoteinstellungsseite zeigt wichtige Informationen zu einem ausgewählten Depot an. So werden Statusinformationen angezeigt, ob das Depot geöffnet, aufgebrochen oder defekt ist, oder auch, ob ein RFID-Tag in einem Depot enthalten ist und ob dieses korrekt ist.
Des Weiteren bietet die Seite Aktionen an, die der Benutzer für dieses Depot bzw. für alle Depots ausführen kann. So kann er Depots schließen, wenn sie offen sind, sie aufbrechen oder einen Dallas einlegen.

Es gibt drei Möglichkeiten die Depoteinstellungsseite nach einem Klick auf ein Depot anzuzeigen:

1. als *Popup*, welches sich als neues Fenster öffnet und später wieder geschlossen werden muss
2. als *Frame*, welches sich neben der Darstellungsseite öffnet
3. als *per Javascript nachgeladener HTML-Code*, welcher an beliebiger Stelle angezeigt werden kann

4.1.3.1 Popup

Eine Darstellung als Popup ist für den Benutzer eine Belastung. Es öffnet sich als eigenständiges neues Fenster und muss zum Arbeiten im Vordergrund gehalten werden. Wenn Aktionen vorgenommen werden und deren Auswirkungen auf der Darstellungsseite zeitgleich verfolgt werden sollen, so muss das Popup an eine Position geschoben werden, an der es keine wichtigen Elemente verdeckt. Ein Klick auf die Darstellungsseite befördert das Popup in den Hintergrund. All diese Eigenschaften weisen das Popup als schlechtestes Darstellungsmittel für die Depoteinstellungsseite aus. Der Benutzer erwartet, dass die gewünschten Informationen im selben Browserfenster dargestellt werden in dem er arbeitet. [Vgl. LN08, S. 70]
Popups haben auch einen negativen psychologischen Effekt, der erwähnenswert ist: den Benutzern sind Popups hauptsächlich aus störender und nerviger Werbung bekannt. Eine Darstellung wichtiger Informationen in einem Popup wird ebenfalls damit in Verbindung gebracht und erzeugt eine teilweise negative Bewertung der Software. [Vgl. LN08, S. 71]
Da praktisch in jedem aktuellen Browser Software und Plugins installiert sind, die Popups blockieren, könnte es zudem zu Problemen kommen, weil bei manchen Benutzern das Popupfenster gar nicht erst geöffnet wird. Zwar kann dies meist eingestellt werden, doch dem Benutzer ist dies nicht zuzumuten.

4.1.3.2 Frame

Eine weitere Darstellungsmethode ist das Frame. Es unterscheidet sich vom Popup hauptsächlich dadurch, dass neue Seiteninhalte nicht als eigenständiges Fenster geöffnet werden, sondern als Teil der Webseite geöffnet werden, in dem ein Link geklickt wurde. Auch diese Darstellungsweise hat Nachteile. So existiert das Frame auch dann und beansprucht Platz auf der Webseite, wenn es noch keinen Inhalt gibt, den es anzeigen soll. Dies ist besonders kritisch, wenn die Virtualisierungssoftware auf PCs ausgeführt werden soll, die einen Bildschirm mit geringer Auflösung haben (z.B. SPCs mit Touchscreen, welche eine Auflösung von z.B. 800 x 600 Pixeln haben können).
Frames unterstützen zudem kein Floating. Floating ist eine Darstellungsmethode, bei der Elemente, die nicht mehr nebeneinander Platz haben, sich untereinander anordnen. Da Frames dies nicht unterstützen, ist die Darstellung auf Bildschirmen mit kleiner Auflösung

somit auch problematisch.
Es existiert auch eine Frame-Variante namens IFrame, die Floating unterstützt. [Vgl. MK06, S. 404] Doch auch das IFrame hat, wie das Frame, den Nachteil, dass beim Laden von neuem Inhalt ein Klickgeräusch abgespielt wird, als wäre ein Link angeklickt worden [Vgl. Wen07, S. 392]. Dies mag nur ein minimaler Nachteil sein, doch kann häufiges Nachladen von Inhalten dadurch erheblich stören.

4.1.3.3 Per Javascript nachgeladener HTML-Code

Die Implementierung dieser Methode bedarf einiger Einarbeitung, da neben PHP und HTML auch Javascript als Programmiersprache bekannt sein muss, doch stellt sie die vielseitigste Methode dar. Durch die clientseitig ausgeführten Javascripts [Vgl. Fla07, S. 261] steht eine mächtige Programmiersprache zur Verfügung, die erst dann zum Einsatz kommt, wenn die Webseite bereits fertig zum Webbrowser des Benutzers übertragen worden ist.
Da diese Programmiersprache so mächtig ist, wurden Frameworks entwickelt, die häufig benötigte Funktionen und Abläufe automatisieren und dem Programmierer die Arbeit erleichtern. Beispiele für solche Javascript-Frameworks sind jQuery und Prototype.
Unter Verwendung von Prototype ist es somit möglich, beim Laden der Webseite eines leeres *div*-Tag anzulegen, welches später mit Inhalt gefüllt werden kann: `<div id="content"></div>`. Div-Tags eignen sich generell ausgezeichnet, um HTML-Elemente mit gleicher Formatierung zu gruppieren. [Vgl. Häß03, S. 234] Um dieses Div-Tag nun mit Inhalt zu füllen, ist unter Nutzung des Prototype-Frameworks lediglich der Javascript-Befehl `$("content").update("Hallo Welt!")` nötig. In das Div-Tag könnten auch HTML-Inhalte oder Inhalte die durch PHP-Skripte generiert worden sind, eingesetzt werden.
Per Javascript bzw. AJAX lassen sich praktisch alle Elemente einer Webseite dynamisch verändern. So können sich einzelne Webseitenteile regelmäßig aktualisieren, ohne die komplette Webseite neu zu laden (siehe Abschnitt 5.3).
Nachteil der Nutzung von Javascripts ist jedoch, dass im Browser des Benutzers Javascript aktiviert sein muss. In sehr sicher eingestellten Systemen könnte dies nicht der Fall sein. Standardmäßig ist Javascript jedoch in jedem häufig genutzten Browser wie z.B. Internet Explorer, Firefox oder Opera aktiviert.

4.1.3.4 Fazit

Die Nachteile von Popups wiegen so schwer, dass sie als Anzeigeoption nicht in Frage kommen. Popups beeinträchtigen die Benutzerfreundlichkeit zu stark. Frames bieten zumindest in ihrer I-Frame-Variante einen guten Kompromiss aus Layoutbarkeit und Programmierkomplexität, doch stören im Dauerbetrieb mit Klickgeräuschen. Einzig die Nutzung von Javascript hat keine nennenswerten Nachteile, außer dass die Programmiersprache im Browser des Benutzers aktiviert sein muss. Da die Virtualisierungssoftware nur

firmenintern genutzt werden soll und intern ebenfalls Javascript häufig zur Anwendung kommt, kann bedenkenlos Javascript zum dynamischen Nachladen von Inhalten genutzt werden.

4.2 Datenhaltung: MySQL vs. SQLite

4.2.1 Aufbau und Installation

MySQL unterscheidet sich von SQLite dadurch, dass es ein eigenständiges Datenbankserverprogramm ist. SQLite hingegen ist lediglich „eine Sammlung von PHP-Funktionen, die Dateien mit Daten über SQL-Befehle manipulieren." [Gün04, S. 257]
Um MySQL zu verwenden, muss erst die Serversoftware installiert werden. Wo diese Software installiert wird, also ob lokal auf demselben Server, auf dem auch die spätere Anwendung laufen soll oder auf einem anderen Server, ist dabei die Entscheidung des Administrators. Wichtig ist, dass auf dem Server, auf dem die Software später laufen soll, auch die Zugriffsbibliotheken installiert werden, die einen Zugriff auf die MySQL-Datenbank erlauben (egal wo sich diese befindet). Die Verbindung zum jeweiligen Server kann dann mit *mysql_connect($hostname, $benutzername, $passwort)* hergestellt werden. [Vgl. KÖ10, S. 410]
Nach Günther [Gün04, 257 ff.] ergeben sich hingegen für SQLite folgende Punkte:

- Die Installation von SQLite ist wesentlich einfacher als die von MySQL. Es wird bereits in der PHP-Distribution ab Version 5 mitgeliefert und ist dann auch bereits aktiv.
- SQLite *muss* auf demselben Rechner installiert werden, auf dem auch die Software später laufen soll, da die Zugriffe dateibasiert ablaufen und keine Netzwerkverbindung hergestellt wird. Das bringt einerseits den Vorteil geringerer Latenzen mit sich, andererseits aber auch den Nachteil, dass der Webserver nicht vom Datenbankserver trennbar ist. Dafür ist dann auch keine Benutzerauthentifizierung mehr nötig (und wird von SQLite auch nicht unterstützt). [Vgl. Gün04, S. 261]
- Mit externen Programmen auf die SQLite-Datenbank zuzugreifen wird schwieriger. Dies ist besonders in Wartungssituationen ein großer Nachteil, denn da SQLite dateibasiert arbeitet, wird die Datenbank bei jedem Lese- und Schreibzugriff kurz gesperrt. Dadurch sind auch parallele Lese- und Schreibzugriffe erschwert. [Vgl. GBR05, S. 208]

4.2.2 Lizenzen

SQLite und MySQL besitzen unterschiedliche Lizenzen. So wird SQLite als Public Domain zur Verfügung gestellt, gewährt also uneingeschränkte Nutzungsrechte. [Vgl. Kra04, S. 284] MySQL hingegen wurde unter einer dualen Lizenz herausgegeben mit GPL und

einer kommerziellen Lizenz. [Vgl. Mau09, S. 310]
Diese Lizenzen sind jedoch im Fall der Virtualisierungssoftware nicht von Bedeutung, denn sie haben erst dann eine Auswirkung, wenn die Datenbanksysteme selbst durch Programmcode verändert oder erweitert werden. Sie zu nutzen, erfordert keine Lizenzierung.

4.2.3 Geschwindigkeit

Geschwindigkeitsaspekte sind bei der Virtualisierungssoftware kein entscheidender Faktor. Wie bereits erwähnt in Unterabschnitt 4.2.1, beherrscht SQLite prinzipbedingt keine parallelen Lese- und Schreibzugriffe. Dieser Aspekt ist vernachlässigbar, denn lediglich bei der Konfiguration der virtuellen Box wird einmal in die Datenbank geschrieben und sonst fast ausschließlich lesend auf die Datenbank zugegriffen um Statusinformationen der Depots abzurufen. Nur selten, nämlich wenn der Benutzer Aktionen ausführt, wird schreibend zugegriffen. Aufgrund der wenigen Schreibzugriffe wird es deshalb fast nie zu Wartezeiten kommen, wodurch SQLite in etwa gleich schnell wie MySQL ist. [Vgl. GBR05, S. 208]

4.2.4 Fazit

MySQL und SQLite sind bereits vom Arbeitsprinzip her sehr unterschiedliche Datenbanksysteme. MySQL ist gut geeignet zum Betrieb auf verteilten Servern, SQLite ist eine Lösung für eher kleine Projekte, die auf einem lokalen System betrieben werden und die keine häufigen parallelen Lese-/Schreibzugriffe benötigen. Aus diesem Grund fällt die Wahl des Datenbanksystems auf SQLite, denn die Virtualisierungssoftware wird zwar häufig Statusinformationen von Depots abfragen, diese aber nur selten verändern.

4.3 Implementierung des Service

Wie in Unterabschnitt 3.2.3 beschrieben, müssen einige Verhaltensweisen von PHP abgestellt werden, um einen Service zu erhalten, der auch dann weiterläuft, wenn der Benutzer seinen Browser schließt, in dem der Service angestoßen worden ist. Um dies zu gewährleisten, werden die Lösungen aus Unterunterabschnitt 3.2.3.3 und Unterunterabschnitt 3.2.3.4 eingesetzt:

```
set_time_limit(0);
ignore_user_abort(true);
```

Listing 4.1: protocol.php, Ungewollten Benutzerabbruch verhindern

Des Weiteren wird eine Abbruchmöglichkeit implementiert:

```
if (isset($_REQUEST["STOP"]))
{
  $Service->sendSTOP();
  die("Send Stop-Command!");
}
```

Listing 4.2: protocol.php, Gewollten Benutzerabbruch ermöglichen

Um den Serviceabbruch zu initiieren, müssen vom Benutzer an die protocol.php Daten übermittelt werden per GET- oder POST-Parameter, denn $_REQUEST ist ein superglobales Array, welches automatisch alle GET- und POST-Parameternamen und ihre Werte enthält [Vgl. SW08, S. 47]. Zusätzlich enthält es noch die Werte der Superglobalen $_COOKIE, doch einen Wert als Cookie zu übergeben ist keine gängige Praxis, denn in Cookies werden Daten durch die Webseite gespeichert und nicht durch den Benutzer. [Vgl. Nas07, S. 354] Da das Senden von GET-Parametern einfach über den Browser vorzunehmen ist (siehe Unterabschnitt 4.4.1, bietet sich dieses für den Benutzer an. Er ruft deshalb die protocol.php in seinem Browser mit einem angehängten ?STOP auf.
Die Funktion *sendSTOP()* trägt daraufhin einen Abbruchwert in der Datenbank ein, der von dem Dienst bei jedem Schleifendurchlauf kontrolliert wird. Das Vorgehen ist analog zu Listing 3.7, nur dass nicht in eine Datei, sondern in die Datenbank geschrieben wird (und auch von dort gelesen wird).

4.4 Datenübertragung übers Netzwerk

4.4.1 Allgemeine Erklärung von GET-Parametern

Daten werden zwischen WinkeyNet, Anubis und der Virtualisierungssoftware auf einem denkbar einfachen Weg übertragen: per GET-Parameter. Dies vereinfacht die Entwicklung der Virtualisierungssoftware enorm, denn es ist keine Auseinandersetzung mit verschiedenen Internet- und Transport-Protokollen nötig. Durch die Übertragung als GET-Parameter übernimmt die Steuerung der Datenübertragung der Apache Webserver.
Diese einfache Art der Datenübertragung ist möglich, da alle Programmteile aus PHP-Skripten bestehen. Die Verarbeitung der eingehenden Pakete übernimmt der Apache Webserver und PHP stellt anschließend die Daten in einem superglobalen Array bereit. Ein sehr einfaches Beispiel kann diese Art der Datenübertragung verdeutlichen:

```
<?php
  if (isset($_GET['passwort']) && $_GET['passwort'] == "test")
    echo "Passwort korrekt!";
  else
    echo "Passwort falsch!";
?>
```

Listing 4.3: parameter.php zur Darstellung von per GET-Parameter übertragenen Daten

Angenommen dieses PHP-Skript ist nun unter folgender URL erreichbar: http://localhost/parameter.php, so können ihm folgendermaßen Daten per GET-Parameter übergeben werden: http://localhost/parameter.php?passwort=daten. [Vgl. PM05, S. 166] Das Fragezeichen weist Apache an, dass nachfolgend Parameter übergeben werden. Der Name des Parameters steht vor dem Ist-Gleich-Zeichen, der Wert danach. Im Falle mehrerer Parameter werden diese durch ein Ampersand (&) getrennt.
Wird das Skript aus Listing 4.3 mit etwas anderem als http://localhost/parameter.php ?passwort=test aufgerufen, so wird „Passwort falsch!“ ausgegeben, ansonsten „Passwort korrekt“. Die Prüfung mit *isset* sorgt dafür, dass kein Vergleich des Inhalts von $_GET['passwort'] vorgenommen wird, wenn diese Variable gar nicht existiert. Je nach Einstellung des Webservers würde sonst eine Warnung ausgegeben.

4.4.2 Verwendung in der Virtualisierungssoftware

Um von einem Server zum anderen Daten zu senden ist es somit lediglich nötig, eine Empfangsseite, hier *receive.php*, aufzurufen und ihr ein paar GET-Parameter zu übergeben. Es gibt zwei Möglichkeiten des Aufrufs:

1. den direkten Aufruf über einen Link, sodass die jeweilige Webseite geladen wird, die aufgerufen wurde
2. den indirekten Aufruf, wenn zwar ein Aufruf stattfinden soll, der Benutzer davon jedoch keine Kenntnis erlangen soll

Da die Virtualisierungssoftware im Hintergrund Daten übertragen soll, ohne dass der Benutzer sich durch Webseiten bewegen oder Links klicken muss, wird für die Kommunikation zwischen den Softwareteilen die Methode aus Punkt 2 verwendet.
Hierfür bietet PHP die Funktion fsockopen() an. Dieser Befehl baut eine Socketverbindung zu einem Host (identifiziert per IP, Hostname oder Unix Socket) auf. Ein Beispiel:

```
<?php
  $fo = fsockopen("127.0.0.1", 80, $errno, $errstr, 10);
  if ($fo) {
    fputs($fo, "GET /parameter.php?passwort=falsch HTTP/1.0\r\n" .
               "Connection: close\r\n" .
               "Host: " . $_SERVER['SERVER_ADDR'] . "\r\n\r\n");

    while(!feof($fo))
      echo "Antwort: " . fgets($fo, 1024) . "<br />";

    fclose($fo);
  }
  else
    echo "Fehler aufgetreten. (Fehler #" . $errno . ": " . $errstr .")";
?>
```

Listing 4.4: Verbindung und Datenübertragung per GET-Parameter und fsockopen()

Durch den PHP-Code aus Listing 4.4 wird per *fsockopen()* eine Socketverbindung zum lokalen Host (127.0.0.1) mit dem HTTP-Standardport (80) aufgebaut (Zeile 2). Fehlervariablen werden durch *fsockopen()* im Fehlerfall befüllt: $errno mit der Fehlernummer und $errstr mit der textuellen Fehlerbeschreibung.
Der anschließende Verbindungsaufbau zur parameter.php aus Listing 4.3 erfolgt nach dem HTTP-Protokoll. [Vgl. Mün08, S. 510] Nach dem Senden der Anfrage wird die Antwort eingelesen bis keine Daten mehr zu lesen sind (Zeile 8 - 9) und die Verbindung geschlossen (Zeile 11). Sollte ein Fehler aufgetreten sein, sodass $fo *false* ist, so werden die Fehlervariablen ausgegeben (Zeile 14).
Mit diesem Verfahren können an beliebige Skriptdateien Daten per GET-Parameter gesendet und deren Antworten ausgewertet werden. So gibt das obige Beispiel aus Listing 4.4 als Rückgabe den kompletten HTML-Header sowie als Antwort im letzten Schleifendurchlauf die eigentliche Antwort (den „Content“), die durch parameter.php erzeugt worden ist: „Passwort falsch!“;
Im Falle der Virtualisierungssoftware werden größtenteils Daten zwischen den receive.php-Dateien des zu testenden Anubis, WinkeyNet und der Virtualizing-Bibliothek ausgetauscht.

5 Wichtige Implementierungsdetails

5.1 Änderung der IP-Einstellungen des Computers

Eine der gewünschten Funktionen war, die IP-Einstellungen (IP, Subnetzmaske, Standardgateway) der virtuellen Box über die Konfigurationsseite (siehe Unterabschnitt 4.1.1) vornehmen zu können. Die Lösung dieser Aufgabe ist komplex, denn mit purem PHP und ohne die Nutzung von Funktionen eines bestimmten Betriebssystems, ist diese Funktion nicht realisierbar. Dies lässt sich dadurch belegen, dass in der PHP-Dokumentation kein Befehl zum Ändern der IP-Adresse und anderer IP-Einstellungen des Computers existiert. [Vgl. Ach+10]
Deshalb wurde festgelegt, dass die Virtualisierungssoftware ausschließlich auf Windowsrechnern ab Windows XP eingesetzt werden sollte. Mit dieser Einschränkung konnten zwei Techniken zum Verändern der IP-Einstellungen implementiert werden, welche beide die Portierbarkeit der Software auf Linux-Systeme erschweren: die Windows Management Instrumentation, kurz WMI und als Alternative das Kommandozeilenprogramm netsh.
Ein weiteres zu lösendes Problem war, dass wenn die Netzwerkeinstellungen verändert worden sind, das Netzwerkinterface diese Einstellungen übernehmen muss, damit diese wirksam werden. Während dieser Einstellungsübernahme gibt es einen kurzen Verbindungsabbruch, der überbrückt werden muss.
Um nun die Einstellungen vorzunehmen, kann der Benutzer in der GUI eine neue IP, Subnetzmaske und ein neues Standardgateway eingeben (siehe Unterabschnitt 4.1.1).

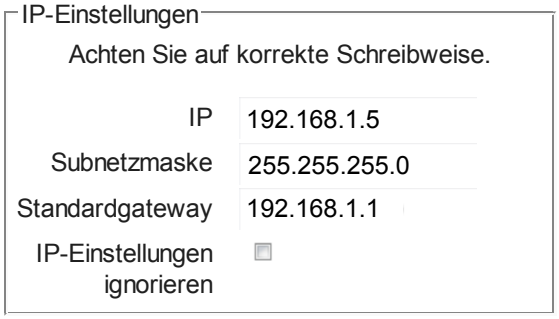


Abbildung 5.1: IP-Einstellungen auf der Konfigurationsseite

Wie diese Daten weiterverarbeitet werden, hängt davon ab, ob der Benutzer auf seinem Computer ein funktionierendes WMI-System hat. Dies wird in Unterabschnitt 5.1.1 festgestellt. Sollte der Benutzer keine Änderungen vornehmen, werden auch keine Vorgänge zur Neukonfiguration der IP-Einstellungen eingeleitet.

5.1.1 Test, ob der Computer WMI unterstützt

Der erste Schritt beim Übernehmen neuer IP-Einstellungen, bestehend aus IP, Subnetzmaske und Standardgateway, ist, zu überprüfen, ob das Hostsystem WMI unterstützt. Auf manchen Computern kann der WMI-Dienst nicht gestartet sein. [Vgl. Ms0, „Erneutes Starten des WMI-Dienstes“] Auch andere Teile des WMI-Systems können beschädigt sein und WMI unzugänglich machen. [Vgl. Ms0, Abschnitt „Erneutes Erstellen des Repositorys“ ff.]
Ein einfacher Test, ob WMI zur Verfügung steht, ist, eine Anfrage an WMI zu senden. Dies geschieht mittels dem Kommandozeilenprogramm *wmic* (WMI Console). So kann aus der WMI-Datenbank, auch Repository genannt, der Name des Betriebssystems abgefragt werden. Der Befehl dazu lautet *wmic os get caption*. Um festzustellen, welcher Text auf den jeweiligen Betriebssystemen zurückgegeben wird, wurde der Befehl versuchsweise auf je einem PC mit einem solchen Betriebssystem ausgeführt. Die Windows Server Versionen wurden dem MSDNAA-Programm der Hochschule Mittweida (FH) entnommen.

- Windows XP: Microsoft Windows XP
- Windows Vista: Microsoft®Windows VistaT
- Windows 7: Microsoft Windows 7
- Windows Server 2003: Microsoft(R) Windows(R) Server 2003
- Windows Server 2008: Microsoft®Windows Server®2008

Nach den jeweiligen Bezeichnungen listen die Betriebssysteme meist noch ihre lizenzierte Version (Standard, Professional, etc.) oder ihr Service Pack (Service Pack 1, Service Pack 2, etc.) auf. Für diese Betrachtung sind diese Ergänzungen jedoch nicht relevant, da die oben genannten „Kernbezeichnungen“ der Betriebssysteme ausgewertet werden sollen.
Ist der Dienst der Windows-Verwaltungsinstrumentation deaktiviert oder kann nicht gestartet werden, so lautet die Ausgabe:

> FEHLER: Beschreibung = Der angegebene Dienst kann nicht gestartet werden. Er ist deaktiviert oder nicht mit aktivierten Geräten verbunden.

Anhand dieser Feststellungen kann nun ein Skript entwickelt werden, welches auf einen vorhandenen, korrekt funktionierenden WMI-Dienst testet, indem es *wmic os get caption* ausführt und anschließend den zurückgegebenen String nach der Zeichenkette „Windows“ durchsucht. Denn alle korrekten Rückgaben enthielten diese Zeichenkette, während der Fehlertext diese nicht enthielt. Die aus diesen Denkansätzen entwickelte Funktion ist diese:

```
1 <?php
    private function isWmiPresent()
3   {
      $output = array();
```

```
5     $returnValue = 0;

7     exec('wmic os get Caption', $output, $returnValue);

9     if ($returnValue == 0) {
        foreach ($output as $line) {
11        if (strpos($line, "Windows") !== false)
            return true;
13      }
      }
15
      $_SESSION['information'][] =  "WMI ist auf diesem PC nicht verfügbar.";
17    return false;
    }
19  ?>
```

Listing 5.1: Test auf WMI-Unterstützung, enthalten in class.ipConfiguration.php

Mit *exec()* (Zeile 7) werden unter PHP beliebige Befehle ausgeführt, die auch in der Kommandozeile (das Windowsprogramm „cmd“) funktionieren würden. So können beliebige Programme auf dem betreffenden Host gestartet werden und auch Befehle der Kommandozeile genutzt werden (z.B. dir). [Vgl. Hud06, S. 99] Die Bildschirmausgaben des ausgeführten Programms werden in die Variable $output geschrieben und der Rückgabewert in die Variable $returnValue.
Das Kommandozeilenprogramm *wmic* gibt als Rückgabewert im Erfolgsfall 0 zurück, bei Hinweisen 1, bei Warnungen 2 und bei Fehlern 3. [Vgl. Ms1]. Die Auswertung dieses Rückgabewerts findet in Zeile 9 statt. Alle Rückgabewerte außer 0 werden als Fehler interpretiert. Im Falle des Rückgabewerts 0 wird mit *foreach()* das Array $output in einzelne $line-Variablen zerlegt und diese nach dem Wort „Windows“ durchsucht (Zeile 10 und 11). Kommt das Wort an einer Stelle des Strings in $line vor, gibt *strpos()* den Index der Fundstelle zurück. Da dieser durchaus auch 0 sein kann (das Wort kommt direkt am Anfang des Strings vor), muss mit dem Identitätsoperator (=== bzw. !==) der Vergleich vorgenommen werden, damit ein Fundindex von 0 nicht als „false“ interpretiert wird. Dies wird dadurch gewährleistet, dass der Identitätsoperator im Vergleich zum Gleichheitsoperator (== bzw. !=) nicht nur den Wert (bei 0 und false in beiden Fällen Null), sondern auch den Typ des Werts vergleicht. [Vgl. Sta04, S. 36] Da die Zahl den Typ *int* hat und false den Typ *boolean*, wird somit bei einem Fund des Teilstrings direkt am Anfang des Strings kein Abbruch hervorgerufen (in Zeile 11) und die Funktion *isWmiPresent()* gibt true zurück.
Sofern *return true* in Zeile 12 ausgeführt wurde (also der if-Zweig in Zeile 11 betreten wurde) wird kein weiterer Code mehr ausgeführt. [Vgl. Kra04, S. 165] Wurde dieser if-Zweig für alle Schleifendurchläufe nicht betreten, wird in die Sessionvariable „Information“ ein Informationstext eingefügt und der aufrufenden Funktion *false* zurückgegeben (Zeilen 16 - 17).

5.1.2 IP-Konfiguration ändern mittels WMI

Praktisch alle Systemeinstellungen eines Windows Betriebssystems ab Windows XP lassen sich über WMI auslesen oder verändern. Dies schließt z.B. auch Listen mit laufenden Prozessen und Diensten mit ein. WMI erlaubt zudem auch das Auslesen der Einstellungen von Netzwerkinterfaces, sowie das Verändern dieser Einstellungen. [Vgl. SRJ08, 876 f.] Nachdem, wie in Unterabschnitt 5.1.1 beschrieben, getestet wurde, ob WMI unterstützt wird und diese Prüfung positiv ausgefallen ist, kann WMI verwendet werden, um die IP-Konfiguration zu verändern.

```
1  private function setConfigurationUsingWMI($newSettings)
   {
3    $com = new COM('winmgmts:{impersonationLevel=impersonate}//./root/cimv2');

5    if (is_object($com)) {
       $configuration = $com->execquery(
7        "SELECT Description FROM Win32_NetworkAdapterConfiguration WHERE IPEnabled
             = 'TRUE'"
       );
9
       if ($configuration->count > 0) {
11       foreach ($configuration as $row) {
           $parameters = $row->EnableStatic(array($newSettings['ip']), array(
               $newSettings['mask']));
13         if (!empty($newSettings['gateway']))
             $row->SetGateways(array($configuration['gateway']));
15         break; //nur für erstes Device
         }
17     } else {
         $_SESSION['errors'][] = "Es konnte kein IP-Interface gefunden werden.";
19        header("Location: index.php?action=configuration");
       }
21
       $com = null;
23
     } else {
25      $_SESSION['errors'][] = "Es trat ein Fehler beim Initialisieren von WMI auf,
            obwohl eine WMI-Verbindung möglich war.";
        header("Location: index.php?action=configuration");
27   }
   }
```

Listing 5.2: IP-Einstellungen ändern mittels WMI, enthalten in class.ipConfiguration.php

In Abschnitt 5.1 wurde beschrieben, dass das Skript ausschließlich auf Windowssystemen lauffähig sein soll. Einer der Gründe, warum dies so ist, liegt in Zeile 3 von Listing 5.2. Dort wird ein COM-Objekt konstruiert, dem ein Moniker, ein String zur Objektidentifizierung, übergeben wird. Obwohl es COM-Objekte durchaus auch unter anderen Betriebssystemen als Windows gibt [Vgl. MS04, S. 1068], ist die ProgID WinMgmts für die Windows-Verwaltungsinstrumentation nur unter Windows verfügbar.
Der Moniker besteht laut Schwichtenberg [Sch10, S. 535] aus:

- der ProgID WinMgmts
- einer optionalen Sicherheitseinstellungsdefinition

- einer optionalen Lokalisierungsangabe und
- dem WMI-Pfad, der nur in wenigen Sondernfällen weggelassen werden darf.

Für den WMI-Pfad dürfen Slashes (/) oder Backslashes (\) verwendet werden und der Name des lokalen Computers darf durch einen Punkt (.) ersetzt werden. [Vgl. Sch10, S. 535]
Nach diesen Regeln entsteht der Objektidentifikationsstring „winmgmts:impersonationLevel=impersonate//./root/cimv2“. Zu erkennen sind die ProgID „winmgmts“, die Sicherheitseinstellungsdefinition „impersonationLevel=impersonate“, die Lokalisierungsangabe „//.“ für den lokalen PC und der WMI-Pfad „/root/cimv2“. [Vgl. SRJ08, S. 883]
Die hier verwendete Sicherheitseinstellungsdefinition bestimmt, dass WMI den Befehl mit den Benutzerrechten ausführt, die das aufrufende Objekt hat. [Vgl. LM01, S. 127] In diesem Fall ist das aufrufende Objekt Apache, der, wenn er als Dienst gestartet ist und sofern dies nicht explizit anders eingestellt worden ist, unter dem Benutzer „SYSTEM“ läuft. Befehle an WMI werden somit ebenfalls mit den Benutzerrechten des Benutzers „SYSTEM“ ausgeführt. So werden Probleme vermieden die auftreten könnten, wenn ein Benutzerprofil mit eingeschränkten Rechten verwendet wird, denn der Apache-Dienst läuft davon unabhängig.
Nach dem das COM-Objekt erstellt worden ist, erfolgt in Zeile 5 eine Prüfung, ob es sich bei $com tatsächlich um ein Objekt handelt. Ist dies nicht der Fall, so ist bei der Erstellung des COM-Objekts ein Fehler aufgetreten. In Zeile 25 und 26 wird deshalb die Sessionvariable „errors“ mit dem Fehlertext befüllt und eine Umleitung zurück auf die Konfigurationsseite durchgeführt.
War die vorangegangene Prüfung des COM-Objekts erfolgreich, so wird über eine SQL-ähnliche Sprache namens WQL [Vgl. Sch10, S. 531] eine Anfrage an das COM-Objekt gestellt (Zeile 6 - 8). Diese Anfrage ermittelt die Namen der Netzwerkadapter im lokalen System (*SELECT Description FROM Win32_NetworkAdapterConfiguration*), jedoch nur für die Adapter, die zum Zeitpunkt der Anfrage aktiviert sind (*WHERE IPEnabled = 'TRUE'*). [Vgl. SRJ08, S. 1095]
In Zeile 10 von Listing 5.2 wird festgestellt, ob mindestens ein Datensatz zurückgegeben worden ist. Ist dies nicht der Fall, wird der Inhalt des if-Blocks übersprungen und Zeile 18 und 19 füllen die Sessionvariable „errors“ und leiten den Benutzer auf die Konfigurationsseite um. Wurde mindestens ein aktiviertes Netzwerkinterface gefunden, so beginnt das Setzen der neuen Einstellungen. Dafür wird in Zeile 12 mit der Funktion *EnableStatic()* für das erste Netzwerkinterface eine statische IP-Einstellung aus IP-Adresse und Subnetzmaske festgelegt. [Vgl. Sch10, 598 f.] Diese Informationen kommen aus dem Array $newSettings, welches der Funktion übergeben worden ist. Es besitzt benannte Indizes mit den Namen „ip“, „mask“ und „gateway“, unter welchen die jeweiligen Informationen abrufbar sind. Da es bei Computern ohne Internetzugang durchaus eine gängige Konfiguration ist, das Standardgateway leer zu lassen (denn über dieses werden Daten aus dem lokalen Netzwerk in ein anderes Subnetz weitergeleitet), ist dieses optional und wird nur geändert, wenn es im $newSettings-Array übergeben worden ist (Zeile 13 und 14).

Aus Gründen der Komplexitätsreduzierung wird davon ausgegangen, dass der Benutzer nur die IP-Einstellungen eines einzigen Netzwerkinterfaces verändern möchte und dass er nur ein Standardgateway besitzt. Sollten mehrere Netzwerkinterfaces existieren, so werden die Einstellungen für das erste Netzwerkinterface übernommen, das von dem WQL-Statement zurückgegeben wurde. Die Schleife mit *foreach* in Zeile 11 wird nach dem ersten zurückgegebenen Gerät in Zeile 15 mit *break* verlassen, um nicht mehreren Netzwerkinterfaces dieselben IP-Einstellungen zuzuweisen.
Des Weiteren ist es nur möglich ein einzelnes Standardgateway anzugeben und nicht mehrere, obwohl die Funktion *SetGateways()* durchaus auch ein Array mit mehreren Gateways unterstützen würde. [Vgl. Sch10, S. 599]

5.1.3 IP-Konfiguration ändern mittels netsh

Netsh ist ein Kommandozeilenprogramm, welches mit Windows Betriebssystemen mitgeliefert wird. Da es bei einer fehlgeschlagenen Prüfung auf WMI (siehe Unterabschnitt 5.1.1) als Alternative verwendet wird, verhindert dies ebenfalls den Einsatz der Virtualisierungssoftware auf einem anderen Betriebssystem als Windows (siehe Abschnitt 5.1). Netsh kann sämtliche Einstellungen auslesen und verändern, die Netzwerkadapter betreffen, nicht nur auf dem lokalen System, sondern auch auf Remotecomputern. [Vgl. SKTR05, S. 404]
Der Quellcode zur Änderung der IP-Einstellung mittels netsh sieht folgendermaßen aus:

```
  private function setConfigurationUsingNetsh($newSettings)
2 {
    $returnValue = 0;
4   $output = array();

6   exec('netsh interface show interface', $output, $returnValue);
    $lastWord = ""; //bei Ausführen des Befehls ist das letzte Wort der Name der
        Verbindung
8   if ($returnValue != 0) {
      $_SESSION['errors'][] =  "netsh konnte nicht ausgeführt werden. (1)";
10    header("Location: index.php?action=configuration");
    } else {
12    foreach ($output as $line) {
        //rückwärts mit einer Schleife wäre schneller, aber bei so wenigen
            Einträgen unnötig
14      $line = trim($line);
        if (strlen($line) > 0)
16        $lastWord = strrchr($line, " ");
      }
18
      $interfaceName = trim($lastWord); //führendes Leerzeichen wegschneiden
20
      $output = array();
22
      //netsh interface ip set address "LAN-Verbindung" static IP-Adresse
          Subnetzmaske Gateway Wert
24    if (empty($newSettings['gateway']))
        $newSettings['gateway'] = "none";
26    else
        $newSettings['gateway'] .= " 1"; //GWMetric hinzufügen
28
      exec(
```

```
30          'netsh interface ip set address "' . $interfaceName . '" static ' .
                $newSettings['ip'] . ' ' . $newSettings['mask'] . ' ' . $newSettings['
                gateway'], $output, $returnValue
        );
32
        if ($returnValue != 0) {
34        $_SESSION['errors'][] = "netsh konnte nicht ausgeführt werden. (2)";
          header("Location: index.php?action=configuration");
36      }
      }
38  }
```

Listing 5.3: IP-Einstellungen ändern mittels netsh, enthalten in class.ipConfiguration.php

Um die IP-Konfiguration eines Netzwerkinterfaces zu verändern, ist es auch in dieser Variante notwendig, den Namen des Netzwerkinterfaces zu kennen. Er lässt sich mittels „netsh interface show interfaces“ ermitteln (Zeile 6). Nach Ausführung dieser Codezeile ist in der Variable $output die Konsolenausgabe des Befehls enthalten, sowie in der Variable $returnValue dessen Rückgabewert. In Abbildung 5.2 ist die Konsolenausgabe dieses Befehls zu sehen, wenn er über die Kommandozeile ausgeführt wird.

```
C:\netsh interface show interface

Verw.-status  Status      Typ          Schnittstellenname
-------------------------------------------------------------------------------------------------------------------
Aktiviert     Verbunden   Dediziert    LAN-Verbindung

C:\>_
```

Abbildung 5.2: Konsolenausgabe des Befehls netsh

Unabhängig von der eingestellten Sprache des Betriebssystems ist somit das letzte Wort der Rückgabe der Name des letzten Netzwerkinterfaces. Wie in Unterabschnitt 5.1.2 gilt auch hier zur Verringerung der Implementierungskomplexität die Annahme, dass ein Computer, auf dem die Virtualisierungssoftware eingesetzt werden soll, nur ein Netzwerkinterface besitzt. Es folgt eine Überprüfung in Zeile 8, ob der Rückgabewert des Befehls ungleich „0“ war. Ein Rückgabewert von „0“ gibt an, dass kein Fehler aufgetreten ist. Ist der Rückgabewert nicht „0“, so wird der else-Zweig (Zeile 35 und 36) ausgeführt, die Sessionvariable „errors“ mit dem Fehlertext befüllt und der Benutzer auf die Konfigurationsseite umgeleitet.
War der Rückgabewert „0“, so wird in einer Schleife mit *foreach()* die Variable $output in einzelne $line-Variablen zerlegt und führende oder nachfolgende Leerzeichen mit *trim()* abgeschnitten (Zeile 12 - 14). Ist die Zeile nicht leer (Zeile 15), so wird mit der Funktion *strrchr()* die Variable $line vom letzten Buchstaben beginnend nach einem Leerzeichen durchsucht. Sobald dieses Leerzeichen gefunden wurde, gibt *strrchr()* den String inklusive der ersten Fundstelle zurück, was dem letzten Wort in dieser Zeile mit einem führenden Leerzeichen entspricht. Sobald dies für die letzte Zeile durchgeführt worden ist, die

mindestens ein Zeichen lang ist, wurde das letzte Wort in der letzten Zeile in $lastWord gespeichert. Somit wurde der Name des letzten Netzwerkinterfaces ermittelt. Mit *trim()* wird in Zeile 19 das führende Leerzeichen entfernt.
Da nun ein neuer Befehl ausgeführt werden soll, wird in Zeile 21 das Array $output wieder mit einem leeren Array initialisiert.
Wie auch in Unterabschnitt 5.1.2 ist es in dieser Funktion möglich, kein Standardgateway im Array $newSettings zu übergeben (z.B. wenn keine Internetverbindung besteht und somit keine Datenübermittlung in ein anderes Subnetz nötig ist). Auch hier besteht das Array aus Elementen mit benannten Indizes mit den Namen „ip“, „mask“ und „gateway“. In Zeile 24 erfolgt die Überprüfung, ob die Gateway-Einstellung leer ist. Ist sie leer, so wird in Zeile 25 der Wert der Einstellung mit „none“ ersetzt, ansonsten wird in Zeile 27 „ 1“ an die bestehende Gatewayeinstellung angehangen. Dies ist die Gatewaymetric, die Wertigkeit des Gateways. Da zur Komplexitätsreduzierung hier nur maximal ein Gateway verwendet wird, darf dieses auch die höchste Wertigkeit von 1 erhalten. Für Windows bedeutet dies, dass diese Route die schnellste in das andere Subnetz ist und somit bevorzugt werden sollte. [Vgl. HB05, S. 126]
In Zeile 29 wird schließlich der Befehl zum Ändern der IP-Konfiguration ausgeführt, welcher aus den einzelnen Teilen zusammengesetzt wird. Der zusammengesetzte Befehl lautet dann z.B. *netsh interface ip set address „LAN-Verbindung“ static 192.168.1.6 255.255.255.0 192.168.1.1 1*. Nach Ausführen des Befehls wird die Bildschirmausgabe erneut in $output und der Rückgabewert in $returnValue gespeichert. Anschließend wird in Zeile 33 der Rückgabewert überprüft, ob dieser „0“ ist (kein Fehler aufgetreten) oder nicht. Ist ein Fehler aufgetreten, wird die Sessionvariable „errors“ befüllt und der Benutzer auf die Konfigurationsseite umgeleitet.

5.1.4 Überbrückung der Verbindungsunterbrechung

Bei Änderung der IP-Konfiguration gibt es eine Verbindungsunterbrechung während Windows die Einstellungen übernimmt. Zudem gibt es ein Problem, wenn der Benutzer die Virtualisierungssoftware mit einer festen IP im Browser aufgerufen hat, z.B. http://192.168.1.5/virtualBox. Hat er seine IP nun auf 192.168.1.6 geändert, so erreicht er die Webseite nicht mehr unter der alten Adresse, sondern unter http://192.168.1.6/virtualBox. Damit der Benutzer diesen Wechsel nicht selbst vornehmen muss (und währenddessen eine Webseite im Browser angezeigt bekommt, die ihm mitteilt, dass die Seite nicht gefunden wurde), wird er noch vor dem IP-Wechsel auf eine Webseite umgeleitet, die ihn informiert, dass ein Konfigurationswechsel eine Wartezeit erfordert.

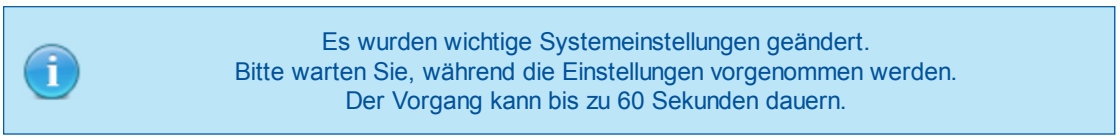


Abbildung 5.3: Hinweis beim Ändern der IP-Konfiguration

Abbildung 5.3 zeigt den Hinweis, der angezeigt wird, wenn die IP-Konfiguration geändert wird. Da diese Seite noch angezeigt wurde, bevor die Änderungen durchgeführt wurden, muss der Benutzer nun auf die Webseite mit der neuen IP umgeleitet werden. Dafür wird das Meta-Element „refresh“ verwendet, welches den Benutzer nach einer gewissen Zeit auf eine andere Webseite weiterleitet. Für die Ausgabe des HTML-Codes für den Hinweis und für die Weiterleitung des Benutzers wird diese Funktion verwendet:

```
  public function getRedirectHtml($successPage, $step)
2 {
    $htmlHelper = new HtmlHelper();
4   //Pfad ermitteln ohne aufgerufene Datei: /virtualizing/validate.php -> /
        virtualizing
    $directory = substr(
6             $_SERVER['SCRIPT_NAME'], 0, strlen($_SERVER['SCRIPT_NAME']) - strlen(
                  strrchr($_SERVER['SCRIPT_NAME'], "/"))
          );
8
    $html = null;
10  $html .= "<html><head>";
    $html .= "<link rel='stylesheet' href='./style/style.css' />";
12
    if ($step == "ip") //neue IP, alter Port
14    $html .= "<meta http-equiv='refresh' content='$this->_delay; URL=http://" .
          $this->_newConfiguration['ip'] . ":" . $_SESSION['apacheConfiguration']->
          getApachePort() . $directory. "/validate.php?step=port&success=
          $successPage'>";
    else if ($step == "port") //neuer Port
16    $html .= "<meta http-equiv='refresh' content='$this->_delay; URL=http://" .
          $this->_newConfiguration['ip'] . ":" . $this->_apachePort . $directory. "
          /index.php?action=$successPage&restoreSettings'>";
    else
18    die("Redirect angefordert, aber der Step ist falsch.");

20  $html .= "</head><body>";
    $html .= "<div id='distance'></div>";
22  $html .= "<div id='container'>";
    $html .= $htmlHelper->htmlInformationBox(array(
24            "Es wurden wichtige Systemeinstellungen geändert.<br />
              Bitte warten Sie, während die Einstellungen vorgenommen werden.<br />
26            Der Vorgang kann bis zu 60 Sekunden dauern."
              ), "information");
28  $html .= "</div>";
    $html .= "</body></html>";
30
    return $html;
32 }
```

Listing 5.4: Ausgabe des Weiterleitungshinweises und Weiterleitung des Benutzers, enthalten in class.redirector.php

Diese Funktion stellt hauptsächlich HTML-Code zusammen. Um den Benutzer weiterleiten zu können auf die neue IP, wird zuerst in Zeile 5 der Pfad ermittelt, in dem sich der Benutzer zuletzt mit seinem Browser befand. Dadurch wird beachtet, dass die Virtualisierungssoftware in verschieden benannten Ordnern installiert sein kann. War der Pfad zur aufgerufenen Datei (die dann diese Funktion aufrief) z.B. „/virtualizing/validate.php“, so liegt dieser Pfad in der Superglobalen $_SERVER im Index SCRIPT_NAME. Die Stringoperation *substr()* führt nun folgendes durch: es wird in diesem Pfad von hinten

das erste Vorkommen eines Slashes (/) gesucht und der Pfad von dort an bis zum Ende als Teilstring verwendet (im Beispiel „/validate.php“). Anschließend wird die Länge dieses Teilstrings bestimmt (13) und von der Länge des Gesamtstrings abgezogen (26 - 13 = 13). Nun ist bekannt wie lang der Teilstring sein muss, der ausschließlich die Pfadnamen und keinen Skriptnamen enthält. Der Teilstring von Index 0 bis Index 13 ist damit: „/virtualizing“.
In Zeile 13 und 15 erfolgt eine Auswertung ob die IP oder der Port geändert worden sind. Dies wird als Parameter an die Funktion übergeben. In dieser Erläuterung wird nur die Änderung der IP behandelt, da die Änderung des Apache-Ports (mit Ausnahme der Funktion, die den Port in der Apache-Konfigurationsdatei ändert) analog erfolgt. Es wird also davon ausgegangen, dass $step mit dem Wert „ip“ übergeben worden ist und $successPage mit dem Wert "draw“. Des Weiteren wurde vor Ausführung dieser Funktion eine Funktion ausgeführt, welche $this->_newConfiguration korrekt mit der neuen IP (als Beispiel wird 192.168.1.6 angenommen), sowie der neuen Subnetzmaske (255.255.255.0) und dem Gateway (192.168.1.1) befüllt hat. Auch wurde $this->_delay auf einen Wert von 15 gesetzt. Der in Zeile 14 generierte HTML-Code ist somit:

```
<meta http-equiv='refresh' content='15; URL=http://192.168.1.6:80/virtualizing/
    validate.php?step=port&success=draw'>";
```

Listing 5.5: Von getRedirectHtml() generierter HTML-Code für den Redirect

Der Benutzer wird dadurch nach 15 Sekunden zum nächsten Schritt, dem Ändern des Apache-Ports, weitergeleitet. Der Apache-Port wird dann in der apache.conf geändert, Apache neugestartet und währenddessen erneut ein Hinweis (siehe Abbildung 5.3) angezeigt, dass Änderungen vorgenommen werden. Dies geschieht analog.
Damit der HTML-Code überhaupt angezeigt wird, während die PHP-Skripte weiter arbeiten (denn während der Hinweis angezeigt wird, müssen die IP- bzw. Apachekonfiguration geändert werden, Sessionvariablen gesichert und wiederhergestellt werden etc.), ist zudem noch ein Trick notwendig. Denn während ein PHP-Skript lädt, erfolgt keine Ausgabe am Bildschirm, sofern diese nicht erzwungen wird. [Vgl. Hud06, S. 185]

```
1  function showRedirectPage($step)
   {
3    global $successPage;
     global $redirector;
5
     ob_end_clean();
7    header("Connection: close\r\n");
     header("Content-Encoding: none\r\n");
9    ignore_user_abort(true);
     ob_start();
11   echo $redirector->getRedirectHtml($successPage, $step);
     $size = ob_get_length();
13   header("Content-Length: $size");
     ob_end_flush();
15   flush();
     ob_end_clean();
17 }
```

Listing 5.6: Trick zur Anzeige des Weiterleitungshinweises, enthalten in validate.php

Die Funktion in Listing 5.6 hat die Aufgabe, diese Ausgabe zu erzwingen. Bei Aufruf leert sie mittels *ob_end_clean()* den standardmäßig vorhandenen Ausgabepuffer, ohne ihn auf den Bildschirm auszugeben (Zeile 6). Anschließend werden Verbindungsparameter für das HTTP-Protokoll übergeben (Zeile 7 und 8). In Zeile 9 wird unterbunden, dass der Benutzer mit einem Klick auf „Abbrechen“ im Browser oder einem Druck auf Escape auf der Tastatur das Skript abbrechen kann (siehe Unterunterabschnitt 3.2.3.4). Zeile 10 startet nun einen neuen Ausgabepuffer mittels *ob_start()*. In diesen Puffer wird in Zeile 11 durch *echo* der HTML-Code aus Listing 5.4 ausgegeben. Zeile 12 und 13 ermitteln die Größe des derzeitigen Ausgabepuffers in Byte und senden diese als Header des HTTP-Protokolls an den Browser des Benutzers. Der Browser hat nun Kenntnis darüber, wieviele Daten ihn erreichen werden. Sobald diese Datenmenge erreicht ist, stellt er die Webseite dar. Mit *ob_end_flush()* wird nun der Ausgabepuffer erneut beendet und gleichzeitig aber auch auf den Bildschirm ausgegeben. Je nach Konfiguration des Webservers kann es zudem dazu kommen, dass auch diese Ausgaben nochmals zwischengespeichert werden. Dies ist der Fall, wenn *implicit_flush* in der PHP-Konfiguration des Servers deaktiviert ist. [Vgl. Wol04, S. 537]. Es wird deshalb in Zeile 15 nochmals ein *flush()* durchgeführt. Sollte *implicit_flush* aktiviert sein, so wird der mittlerweile leere Ausgabepuffer an den Browser gesendet, für den Benutzer geschieht somit keine sichtbare zusätzliche Aktion. War die Option deaktiviert, so wird erst jetzt der Ausgabepuffer tatsächlich geleert und auf den Bildschirm des Benutzers ausgegeben. In Zeile 16 wird der Ausgabepuffer ohne Ausgabe auf den Bildschirm geleert – ab hier gibt es keine sinnvollen Ausgaben mehr, bis die Weiterleitung des Benutzers auf die nächste Webseite erfolgt.
Ein genauer Ablaufplan einer kompletten IP- und Apache-Port-Konfiguration ist in Anhang A im Kapitel 4.1 enthalten.

5.2 Feldvalidierung

In den Produktleistungen in Kapitel 6 in Anhang A wird in /L10/ (Benutzerfreundlichkeit) besagt, dass der Benutzer bei falschen Eingaben Fehler- bzw. Warnmeldungen erhält. Hierfür wurde die Klasse class.inputValidator.php geschaffen (siehe Abbildung 5.4).

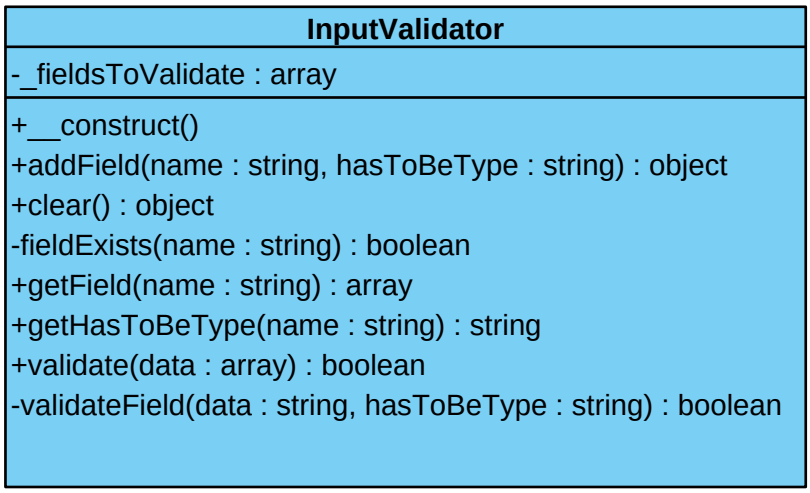


Abbildung 5.4: Klassendiagramm class.inputValidator.php

Nach dem Absenden eines Formulars im Browser werden die Daten (je nach „method“-Attribut der Form, standardmäßig aber als POST-Parameter) an den Server übermittelt. Die nachfolgende Erläuterung geht davon aus, dass die Daten als POST übergeben werden und somit im superglobalen Array $_POST gespeichert sind. Es sollte an jeder Stelle in der Virtualisierungssoftware diese Klasse zur Eingabevalidierung herangezogen werden können.
Um dies zu erreichen, bietet die Klasse HtmlHelper die Funktion *validate()* an (siehe Listing 5.7)

```
1  public function validate($hasToBeType, $name = null)
   {
3    if ($name)
       $_SESSION['inputValidator']->addField($name, $hasToBeType);
5    else
       $_SESSION['inputValidator']->addField($this->_lastFieldName, $hasToBeType);
7
     return $this;
9  }
```

Listing 5.7: Funktion zum Markieren von zu validierenden Feldern, enthalten in class.htmlHelper.php

Mit der HtmlHelper-Klasse werden einfache HTML-Quellcodes wie z.B. für Eingabefelder und Forms generiert. Um ein generiertes Eingabefeld zu validieren, muss der *validate()*-Funktion der Typ übergeben werden, dem der an den Server übermittelte Inhalt dieses Feldes genügen muss. Wird kein Name für das zu validierende Eingabefeld angegeben, so wird der Name des letzten generierten Eingabefeldes verwendet ($this->_lastFieldName, Zeile 6). Um immer dasselbe InputValidator-Objekt zu verwenden und somit zu gewährleisten, dass verschiedene Eingabefelder nicht mit verschiedenen InputValidator-Klassen zu validieren sind, wurde diese zuvor einmalig instanziert und das Objekt in die Sessionvariable „inputValidator“ geschrieben.
Ein Beispielformular mit zwei Eingabefeldern, die validiert werden sollen, könnte folgendermaßen erzeugt werden:

```
1  $html = $htmlHelper->addhtmlInput("text", "karte", "long", "0")->validate("cardno
       ")
                   ->addHtmlInput("text", "name", "long")->validate("required")
3                  ->addHtmlInput("submit", "send", null, "Daten übertragen")
                   ->getHtml();
5  $html = $htmlHelper->htmlEncloseWithForm($html, "kartenleserForm", "validate.php"
       );
   echo $html;
```

Listing 5.8: Erzeugen eines Beispielformulars mit Validierung

Angenommen ein Benutzer gibt in das obere Feld für die Kartennummer ein: „1234“ und in das Feld für den Namen: „Max Mustermann“ und klickt dann auf „Daten übertragen“. Die Daten werden dann an den Server gesendet, welcher im superglobalen Array $_POST diese Daten speichert. Um diese nun validieren zu lassen, ist folgender PHP-Quellcode in validate.php notwendig:

```
if (isset($_POST['send'])) {
  if ($_SESSION['inputValidator']->validate($_POST))
    //Code für Erfolg
  else
    //Code für Fehlschlag
}
```

Listing 5.9: Validieren der übertragenen Daten

Dieses Codestück ist ausreichend um beide Felder zu validieren und im Fehlerfall Maßnahmen ergreifen zu können. Die Funktion *validate()* der Klasse InputValidator, die in Zeile 2 von Listing 5.8 die komplette Validierung übernimmt, soll deshalb ebenfalls erläutert werden:

```
public function validate($data)
{
  $errors = array();

  foreach ($this->_fieldsToValidate as $key => $value) {

    //Spezialfall Checkbox
    if ((!isset($data[$key])) && $this->_fieldsToValidate[$key] == "bool")
      $data[$key] = "0"; //off / false

    if (isset($data[$key])) {
      if (!$this->validateField($data[$key], $this->_fieldsToValidate[$key]))
        $errors[] = $key; //dann z.B. Ausgabe: Keine gültige Eingabe o.ä.
    } else
      $errors[] = $key;
  }

  $_SESSION['data'] = $data;

  if (count($errors) > 0) {
    $_SESSION['validationErrors'] = $errors;
    return false;
  }
  else
    return true;
}
```

Listing 5.10: Validierungsfunktion, enthalten in class.inputValidator.php

Zudem wird zur Erläuterung der Funktion aus Listing 5.10 noch die Funktion *validateField()* (verwendet in Zeile 12) aus der Klasse InputValidator benötigt (hier stark eingekürzt auf die zwei im Beispiel Listing 5.8 verwendeten Validierungstypen):

```
private function validateField($data, $hasToBeType)
{
  $validates = false;

  switch($hasToBeType)
  {
    case "cardno":
      if (strlen($data) <= 20)
        $validates = true;
      break;
```

```
12        case "required":
            if (!empty($data))
14            $validates = true;
            break;
16      }

18      return $validates;
    }
```

Listing 5.11: Validierungsfunktion für einzelne Felder, enthalten in class.inputValidator.php

Wird nun die Validierung verwendet, wie in Listing 5.9 gezeigt, so wird die Funktion aus Listing 5.10 aufgerufen. Das Formular wurde zuvor wie in Listing 5.8 erzeugt und damit das Array $_fieldsToValidate durch die Funktion *validate()* der Klasse HtmlHelper (siehe Listing 5.7) gefüllt. In dem Array befinden sich nun die Elemente „karte“ mit dem Wert „cardno“ sowie „name“ mit dem Wert „required“.
Die Funktion *validate()* der Klasse InputValidator (siehe Listing 5.10) zerlegt nun in Zeile 5 dieses Array, sodass in $key der Name des Elements („karte“ oder „name“) und der jeweilige geforderte Typ (im Beispiel „cardno“ oder „required“) in $value enthalten sind. Der Funktion wird in Listing 5.9 in Zeile 2 als Parameter das superglobale Array $_POST übergeben. Dieses steht nun in Listing 5.10 der Funktion als Parameter $data zur Verfügung.
Zuerst erfolgt in Zeile 8 die Überprüfung auf den Spezialfall, dass es sich um eine Checkbox handelt. Checkboxen übertragen, wenn sie nicht angehakt worden sind, keinen Wert. Wurden sie angehakt, übertragen sie den Wert, der ihrem „value“-Attribut zugewiesen worden ist (in der Virtualisierungssoftware ist dann der „value“ = 1). [Vgl. Gün04, S. 63] Damit nicht angehakte Checkboxen nicht an der Validierung scheitern, wird ihr Wert in Zeile 9 explizit auf 0 gesetzt. Ob es sich um eine nicht-angehakte Checkbox handelt wird daran festgestellt, dass kein Element im Daten-Array unter dem Namen der Checkbox existiert und dass der Wert dieses Elements als Bool(/Boolean) validiert werden soll.
In Zeile 12 folgt schließlich die Überprüfung, ob das Element mit dem in $key enthaltenen Namen des zu validierenden Eingabefeldes auch wirklich im Daten-Array zur Verfügung steht. Falls nicht, so wird der Name des Feldes direkt dem Array $errors hinzugefügt (Zeile 15). In dieses Array werden die Namen der Eingabefelder eingetragen, deren Daten nicht valide sind. Sollte aber ein passendes Element existieren, so wird mit der *validateField()*-Funktion in Zeile 12 überprüft, ob die Daten dieses Elements die Validierungsanforderung erfüllen.
Hierzu muss Listing 5.11 betrachtet werden. Der in diesem Listing beschriebenen Funktion wurden als Parameter $data der zu überprüfende Wert und als Parameter $hasToBeType der Typ übergeben, den $data haben soll. Der *switch* in Zeile 5 wählt den passenden Programmzweig aus und die Überprüfung von $data findet statt. Werden die Validierungsschritte in Zeile 8 bzw. 14 bestanden, so gibt die Funktion *true* an die Funktion *validate()* in Listing 5.10 zurück.
Nachdem von *validate()* alle Elemente von $data überprüft worden sind, werden die Daten

in eine Sessionvariable gesichert, damit diese auch von anderen Skriptteilen benutzt werden können (Zeile 18). Zudem existiert nun ein Array „errors“, das die Namen der Elemente enthält, die keine validen Werte haben. Im angenommenen Beispiel würden „karte“ und „name“ valide Werte besitzen und dementsprechend ist $errors ein leeres Array. Die Zählung der Elemente von $errors in Zeile 20 würde somit 0 ergeben und die Funktion *validate()* würde *true* an Listing 5.9 zurückliefern. Hätte z.B. „karte“ hingegen einen Wert, der mehr als 20 Zeichen lang ist, so würde *validateField()* in Listing 5.11 *false* an *validate()* in Listing 5.10 zurückliefern. Dadurch würde „karte“ an das Array $errors angefügt, das Array hätte eine Größe von einem Element und somit würde in Zeile 20 der if-Zweig betreten. Dann würde $errors in der Sessionvariable „validationErrors“ gespeichert und die Funktion *validate()* würde *false* zurückgeben.
Um diesen Fehler dann lesbar auszugeben, existiert in der Klasse HtmlHelper die Funktion *htmlInformationBox*, welche als Parameter $information die auszugebende Nachricht bzw. im Fall von Validierungsfehler den Namen des nicht-validierenden Elements und als Parameter $type die Art der Informationsbox erwartet. Diese kann „validation“, „error“, „information“, „warning“ oder „success“ sein.
Im Fall der Validierungsfehler erfolgt die Ausgabe dann z.B. mit:

```
1  if (isset($_SESSION['validationErrors'])) {
     $html = $this->htmlInformationBox($_SESSION['validationErrors'], "validation");
3    unset($_SESSION['validationErrors']);
     echo $html;
5  }
```

Listing 5.12: Ausgabe von Validierungsfehlern

Im Falle einer Kartennummer im Element „karte“, die nicht validiert, sähe die Ausgabe (CSS-formatiert) dann folgendermaßen aus:

Abbildung 5.5: Beispiel Validierungsfehler

Die Darstellung in Abbildung 5.5 ist etwas technisch geraten, da sie den Feld- und Validierungsnamen anzeigt, doch reicht dies für den Prototyp aus.
Zusammengefasst muss der Programmierer lediglich seine mit dem HtmlHelper erzeugten Eingabefelder (siehe Listing 5.8) mit *->validate("typ")* ergänzen und den Typ in Listing 5.11 ergänzen, sofern er noch nicht vorhanden ist. In dem Teil seines Skripts in den er nach Absenden des Formulars gelangt, kann er dann mit dem Quellcode aus Listing 5.9 die Validierung vornehmen und den Benutzer entsprechend des Ergebnisses der Validierung umleiten und informieren.

5.3 AJAX-Polling

Wenn die Virtualisierungssoftware von einem Benutzer in Betrieb genommen wurde und er sich auf der Darstellungsseite (siehe Unterabschnitt 4.1.2) befindet, kann er nicht nur Aktionen in der Virtualisierungssoftware durchführen. Es können auch Aktionen in WinkeyNet oder Anubis ausgeführt werden, welche z.B. das Öffnen eines Deptos in der virtualisierten Box nach sich ziehen. Damit der Benutzer nicht nach jeder Aktion in WinkeyNet oder Anubis F5 drücken muss um den Inhalt des Browserfensters zu aktualisieren, ist es sinnvoll, den Inhalt der Darstellung regelmäßig zu pollen.
Um dies zu erreichen wird ein AJAX-Framework namens Prototype eingesetzt, welches in Unterunterabschnitt 4.1.3.3 bereits erklärt worden ist. Prototype ermöglicht es dem Programmierer, den Inhalt eines Div-Elements zu verändern. Der Inhalt kann aktualisiert werden, teilweise verändert oder auch komplett (inklusive des Div-Elements) ersetzt werden.
Damit der Benutzer der Virtualisierungssoftware immer einen möglichst aktuellen Überblick über die Statusinformationen der einzelnen Depots hat, soll die Darstellungsseite (siehe Unterabschnitt 4.1.2) deshalb mittels AJAX-Polling aktuell gehalten werden. Dies soll allerdings so erfolgen, dass z.B. geöffnete Eigenschaftsseiten von Depots nicht geschlossen werden - es soll also lediglich ein teilweises Update der Webseite erfolgen. Prototype bietet dafür die Funktion „PeriodicalUpdater“. Diese aktualisiert einen Div-Container mit einem bestimmten „id“-Attribut in einem bestimmten Zeitintervall (unglücklicherweise „frequency“ genannt) und kann im Falle, dass sich gegenüber dem vorherigen Stand nichts geändert hat, dieses Intervall noch weiter verzögern („decay“).
Damit Prototype funktioniert, muss die Prototype-Javascript-Bibliothek heruntergeladen worden sein und in dem HTML- oder PHP-Dokument, in dem Prototype verwendet werden soll, muss diese Bibliothek im „head“-Element geladen worden sein: `<script type= "text/javascript"src="./javascript/prototype.js"></script>`. Auf der Darstellungsseite, die in class.virtualizingController.php zusammengesetzt wird, müssen zudem die einzelnen Bereiche in verschiedene Div-Container eingeteilt werden:

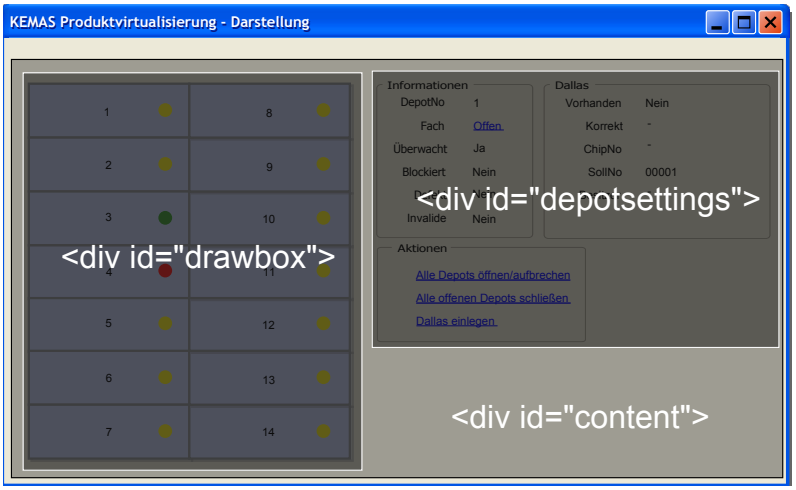


Abbildung 5.6: Div-Container der einzelnen Bereiche der Darstellungsseite

Mittels eines Javascripts kann nun ein PeriodicalUpdater erzeugt werden:

```
$html = "<script type='text/javascript'>";
$html .= "new Ajax.PeriodicalUpdater('drawbox', 'ajax.php?html=drawbox',
          {
            method: 'get',
            frequency: 5,
            decay: 1
          });
        ";
$html .= "</script>";
```

Listing 5.13: Erzeugung eines PeriodicalUpdaters, enthalten in class.virtualizingController.php

Der erste Parameter in Zeile 2, hier „drawbox“ gibt an, welches Div-Element mit den empfangenen Daten aktualisiert werden soll. Der zweite Parameter „ajax.php?html=drawbox“ gibt an, woher die Daten kommen sollen, die in das Div-Element eingefügt werden sollen. Die ajax.php wird durch den PeriodicalUpdater mit einem GET-Request aufgerufen (Zeile 4), in einem Zeitabstand von 5 Sekunden (Zeile 5). Damit die Zeit sich nicht verlängert, wenn keine Änderung erfolgt ist, sondern kontinuierlich alle 5 Sekunden gepollt wird, wird der „decay“ auf 1 gesetzt. Mit diesem Wert wird das Zeitintervall im Falle unveränderten Inhalts sonst multipliziert.
Das Div-Element mit dem „id“-Attribut „drawbox“ wird nun kontinuierlich alle 5 Sekunden aktualisiert, während die anderen Div-Elemente mit den „id“-Attributen „content“ und „depotsettings“ unberührt bleiben.

5.4 Testen des Codes mit Unit Tests

Besonders in großen Softwareprojekten kann nicht immer abgeschätzt werden, ob eine Änderung an einer Klasse oder Funktion nicht nur mittelbare Auswirkungen auf diese Klasse oder Funktion hat, sondern auch Auswirkungen auf andere Klassen und Funktionen, die die geänderten Programmteile ebenfalls verwenden. Auch bei großen Änderungen und Umstrukturierungen müssen eventuell viele Funktionalitäten der Software erst getestet werden. Von Hand geht dies über Tabellen mit Beispiel- und Testwerten zu bewältigen, über Anleitungen, welche Buttons und Aktionen in welcher Reihenfolge auszulösen sind etc. Da so ein Vorgehen zu zeitaufwendig ist, wurden sogenannte Unit Tests entwickelt. Das Prinzip hinter Unit Tests besagt, dass der Programmierer versucht, die Einheiten (Units) einer Software so klein wie möglich zu halten, damit sie einfach automatisiert getestet werden können. Statt großen „göttlichen“ Funktionen, die schwer zu warten und zu testen sind, sollen kleine überschaubare Funktionen und Klassen geschrieben werden, deren Teilergebnisse sich automatisiert überprüfen lassen. Durch dieses Umdenken bei der Programmierung werden hochkomplexe Funktionen vermieden, die später schwer zu warten sind und eventuell sehr fehleranfällig sein können. Im Folgenden werden Funktionen als Units angenommen.

5.4.1 PHPUnit und Testkonfiguration

Für PHP existiert eine Software zur Ausführung von Unit Tests namens PHPUnit. Diese wird in der Kommandozeile von Windows ausgeführt und verwendet im Hintergrund die PEAR-Bibliothek. PHPUnit ist also genau genommen ein PHP-CLI-Skript, also ein in der Kommandozeile abgearbeitetes PHP-Skript.
Für die Tests existiert ein eigenes Verzeichnis in der Ordnerstruktur der Virtualisierungssoftware. Die Tests befinden sich im Unterordner „tests\unit\virtualizing“. Einen Ordner darüber, in „tests\unit“ müssen zudem vor dem Testen noch zwei Dateien angelegt werden. Die erste Datei ist die phpunit.xml, in dieser sind Einstellungen für PHPUnit festgelegt. Existiert sie, wird sie automatisch geladen. Bootstrap.php ist die zweite Datei. In ihr werden Vorbereitungen zum Testen getroffen, z.B. können darin Konstanten definiert werden, Pfade gesucht werden etc.
Um PHPUnit so einzustellen, dass selbst kleinste Fehlerausgaben, auch Warnungen, als Fehler ausgegeben werden, wird die Konfiguration in phpunit.xml folgendermaßen vorgenommen:

```
1  <phpunit bootstrap="bootstrap.php"
     colors="false"
3    convertErrorsToExceptions="true"
     convertNoticesToExceptions="true"
5    convertWarningsToExceptions="true"
     stopOnFailure="false"
7    syntaxCheck="true">

9    <testsuite name="virtualizing">
       <directory>./</directory>
11   </testsuite>
   </phpunit>
```

Listing 5.14: Alle unerwarteten Ausgaben als Fehler anzeigen, enthalten in phpunit.xml

In Zeile 1 wird die bootstrap.php angegeben, die wie oben erläutert, Vorbereitungen für die Tests trifft. Zeile 2 schaltet eine farbige Darstellung ab, da auf der Kommandozeile getestet werden soll und HTML-Farbformatierungen dort stören würden. Damit sämtliche unerwarteten Ausgaben des zu testenden Skriptes als Fehler gewertet werden, stellen Zeile 3 - 5 ein, dass Fehlermeldungen, Anmerkungen und Warnungen als Exception dargestellt werden sollen. Anschließend wird festgelegt, dass auch bei Fehlern der Testablauf weiter verfolgt werden soll (Zeile 6) - so spart man Zeit, da mehrere Fehler auf einmal angezeigt werden können. In Zeile 7 wird eingestellt, dass die PHP-Skripte der Tests vor der Ausführung auf Syntaxfehler untersucht werden sollen. Zeile 9 - 11 erstellen abschließend die sogenannte Testsuite, die Menge an PHP-Skripten für den Test, und geben an, wo die Testdateien zu finden sind. Der Inhalt der bootstrap.php soll nicht weiter erläutert werden, da sie für die Erläuterung der Tests eine untergeordnete Rolle spielt.

5.4.2 Ein Beispieltest mit PHPUnit

Um nun einen Test durchführen zu können, ist eine Test-Datei nötig. Es bietet sich dabei an, die Testdateien nach den Klassen zu benennen, die sie testen. So existiert z.B. für die Klasse SQLiteConnector eine Test-Datei mit dem Namen sqLiteConnectorTest.php (im Unterverzeichnis tests\unit\virtualizing). Da diese Datei im Original über 50 Zeilen PHP-Code enthält, wurde sie an einigen Stellen gekürzt. Sie soll lediglich als Erklärbeispiel dienen.

```
set_include_path(get_include_path() . PATH_SEPARATOR . dirname(__FILE__) . "
    /../../../classes");
require_once 'PHPUnit/Framework.php';
require_once 'class.sqliteConnector.php';
class sqliteConnectorTest extends PHPUnit_Framework_TestCase
{
  protected $_sqLiteConnector = null;
  protected $_conn = null;

  protected function setUp()
  {
    session_start();
    if (file_exists(dirname(__FILE__) . "/test.db"))
      unlink(dirname(__FILE__) . "/test.db");
    $this->_sqLiteConnector = new SQLiteConnector(dirname(__FILE__) . "/test.db");
  }

  protected function tearDown()
  {
    session_destroy();   //[...]
    unlink(dirname(__FILE__) . "/test.db");
  }

  public function testSQLiteConnector()
  {
    $this->assertNotNull($this->_sqLiteConnector, "Kann SQLiteConnector nicht
        instanzieren.");
    $this->assertEquals(count($_SESSION['errors']), 0, "SQLiteConnector-Konstruktor
         ist fehlerhaft.");
    $this->_conn = $this->_sqLiteConnector->getConnection();
    $this->assertEquals(count($_SESSION['errors']), 0, "SQLiteConnector-Konstruktor
         ist fehlerhaft.");
    $this->assertNotNull($this->_conn, "Keine Verbindung vom SQLiteConnector zurück
         erhalten.");
    //[...]
  }
}
```

Listing 5.15: Gekürzter Auszug aus Test-Datei für die Klasse SQLiteConnector, enthalten in sqLiteConnectorTest.php

Eine Test-Datei benötigt dabei laut Zandstra [Zan10, 382 f.] fünf Voraussetzungen:

1. die Einbindung der zu testenden Klasse und des PHP-Unit-Frameworks (Zeile 2 und 3)
2. eine Testklasse, hier sqLiteConnectorTest (Zeile 4)
3. die Funktion *setUp()*, die dem Test vorausgehende Aktionen ausführt (ab Zeile 7)

4. die Funktion *tearDown()*, die nach dem Test „aufräumt“ (ab Zeile 15)
5. sowie mindestens eine Funktion, die mit dem Wort „test“ beginnt - die tatsächliche Testfunktion, hier testSQLiteConnector() (ab Zeile 21)

Zeile 1 sorgt lediglich dafür, dass nach Klassendateien auch im „classes“-Ordner der Virtualisierungssoftware gesucht wird, denn der Standardsuchpfad umfasst lediglich das aktuelle Verzeichnis und (manchmal) ein Include-Verzeichnis im Ordner der PHP-Installation.
In der Funktion *setUp()* werden, wie bereits erwähnt, Vorbereitungen getroffen, damit der Test immer mit den gleichen Ausgangskonditionen starten kann. Da alle Klassen zum Speichern von Fehlern Sessionvariablen verwenden, ist es wichtig, die Sessionverwaltung zu starten. Dies geschieht in Zeile 9 mit *session_start()*. Zudem könnte ein zuvor nicht normal beendeter Test dafür gesorgt haben, dass bereits eine Datei namens test.db im aktuellen Verzeichnis existiert. Ist dies der Fall, so wird sie in Zeile 11 gelöscht, nachdem zuvor in Zeile 10 auf ihre Existenz geprüft wurde. Anschließend wird ein SQLite-Connector-Objekt erstellt und in einer privat deklarierten Klassenvariable (Deklaration gekürzt) gespeichert (Zeile 12). Dadurch wird automatisch auch eine frische Datenbankdatei angelegt.
Es erfolgt nun der Test, indem automatisch die erste (und diesem Fall einzige) Test-Funktion *testSQLiteConnector()* von PHPUnit aufgerufen wird. PHPUnit bietet dem Programmierer verschiedene Funktionen zum Testen der Korrektheit von Funktionsergebnissen an. Diese Funktionen beginnen mit dem Wort „assert“, was auf Deutsch soviel bedeutet wie „bestätigen“. „Bestätige, dass das Objekt $this->_sqLiteConnector nicht gleich *null* ist“, ist dann in diesem Fall „*assertNotNull()*“ und wird in Zeile 21 verwendet. Im Falle, dass das Objekt gleich „null“ ist, wird eine Exception durch PHPUnit ausgelöst und der beschreibende optionale Text aus dem zweiten Parameter der Assert-Funktion ausgegeben. Eine weitere Assert-Funktion wird in Zeile 24 verwendet. Die Funktion *assertEquals()* stellt sicher, dass Parameter eins und Parameter zwei gleich sind und gibt sonst den Text des dritten Parameters zusätzlich zu einer Exception aus.
Als abschließende Funktion führt PHPUnit, nachdem der Test durchlaufen worden ist, die Funktion *tearDown()* aus. In diesem Beispiel werden dadurch die Sessionvariablen gelöscht (Zeile 17), damit diese nicht den Ablauf eines anderen Tests stören und die Datei test.db im aktuellen Verzeichnis gelöscht.

5.4.3 Hudson als grafische Erweiterung

Das Kommandozeilenprogramm PHPUnit bietet zwar alles, was ein Programmierer benötigt um seine Software gut testen zu können, doch fehlt Kommandozeilenprogrammen häufig die Übersichtlichkeit. Grafische Darstellungen entfallen, für sämtliche Auswertungen müssen die korrekten Kommandozeilenbefehle angegeben werden und eventuell wird sogar Potential verschenkt, weil nützliche Funktionen gar nicht bekannt sind.

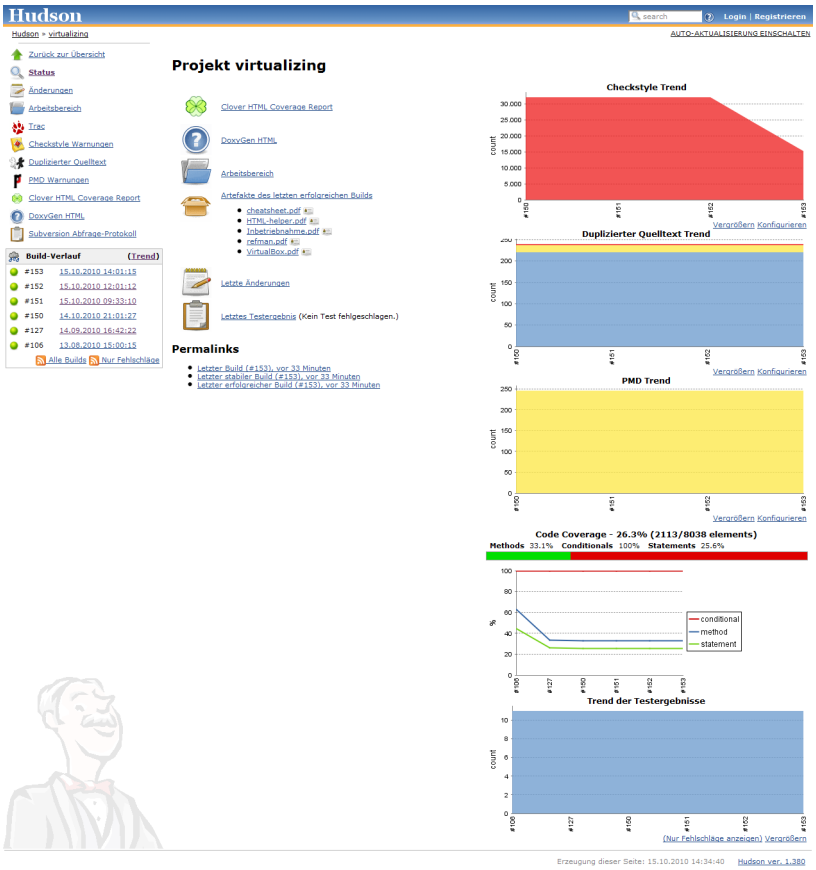


Abbildung 5.7: Startseite von Hudson

Um diese Nachteile zu vermeiden, gibt es die Software „Hudson“. Sie ist nicht nur als grafische Auswertung der PHPUnit-Tests gedacht, sondern führt auch Statistiken über gelungene und fehlgeschlagene Tests. Auch ermittelt die Software die sogenannte Code-Coverage – die Abdeckung des Programmcodes mit Tests in Prozent. Alle Funktionen sind bequem über Links und Buttons erreichbar und Grafiken geben Aufschluss über die letzten Erfolge und Misserfolge.

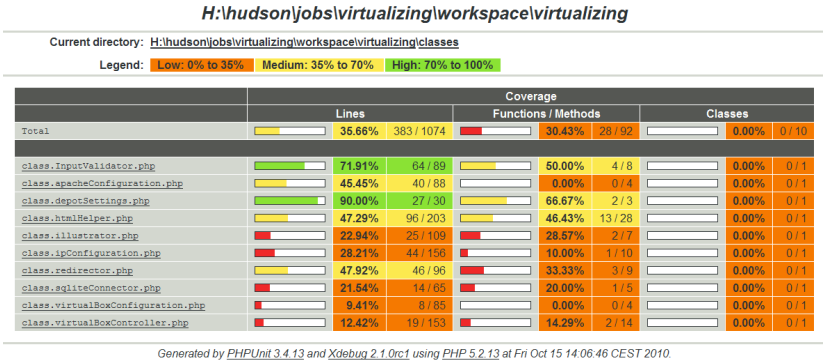
Hudson kann mit Modulen um weitere Funktionen erweitert werden, nicht in das Aufgabengebiet von PHPUnit fallen. So kann Hudson auch den Stil des Programmcodes überprüfen, nach doppeltem Sourcecode suchen, auf übergroße Klassen und Methoden hinweisen sowie die Komplexität von Funktionen und Klassen mit mehreren Algorithmen überprüfen.

In Abbildung 5.7 wird die Startseite von Hudson dargestellt. Auf der linken Seite befindet sich das Navigationsmenü, darunter befindet sich der „Build-Verlauf“. In diesem stehen grüne Symbole für erfolgreich getestete Builds, gelbe Symbole (nicht auf dem Bild zu sehen) für instabile Builds (bringen PHPUnit oder Hudson zum Absturz) und rote Symbole für Builds mit fehlschlagenden Tests.

In der Mitte befinden sich die am häufigsten benötigten Verknüpfungen zur Code-Coverage (Testabdeckung), zu DoxyGen (erstellt aus speziellen PHP-Kommentaren ein

PDF, ähnlich JavaDoc) etc. Hudson ist zudem in der Lage neben den Hauptaufgaben der Software auch zusätzliche Befehle auszuführen, z.B. das Generieren von Latex-Dokumenten. Die daraus resultierenden Dateien werden als „Artefakte“ angezeigt.
Auf der rechten Seite werden Graphen dargestellt zum Zustand der getesteten Software. Zuerst der „CheckStyle“-Graph, welcher die Anzahl der Stilfehler in allen PHP-Dateien anzeigt. Darunter wird der Graph für duplizierten Quelltext dargestellt. Je nach „schwere“ der Duplikation, also je mehr Quelltext dupliziert worden ist, werden die einzelnen Teilgraphen rot, gelb oder blau dargestellt. Der gelbe Graph darunter behandelt sogenannte PMDs, dies umfasst Komplexitätswarnungen, z.B. bei übegroßen Klassen und Funktionen oder Funktionen mit zuvielen Schleifen und Verzweigungen. Der grün-rote Code-Coverage-Graph zeigt an, wieviel Prozent des aktuellen Programmcodes von Tests erfasst und getestet werden. Der Graph darunter stellt die Code-Coverage pro Build dar. Der letzte blaue Graph zeigt an, wie der Testverlauf der letzten Builds war. Bei fehlerhaften Tests wird dort ein roter Balken dargestellt.

5.4.4 Warum nicht 100% Code-Coverage?



H:\hudson\jobs\virtualizing\workspace\virtualizing

Current directory: H:\hudson\jobs\virtualizing\workspace\virtualizing\classes

Legend: Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

	Coverage							
	Lines			Functions / Methods			Classes	
Total	35.66%	383 / 1074		30.43%	28 / 92		0.00%	0 / 10
class.InputValidator.php	71.91%	64 / 89		50.00%	4 / 8		0.00%	0 / 1
class.apacheConfiguration.php	45.45%	40 / 88		0.00%	0 / 4		0.00%	0 / 1
class.depotSettings.php	90.00%	27 / 30		66.67%	2 / 3		0.00%	0 / 1
class.htmlHelper.php	47.29%	96 / 203		46.43%	13 / 28		0.00%	0 / 1
class.illustrator.php	22.94%	25 / 109		28.57%	2 / 7		0.00%	0 / 1
class.ipConfiguration.php	28.21%	44 / 156		10.00%	1 / 10		0.00%	0 / 1
class.redirector.php	47.92%	46 / 96		33.33%	3 / 9		0.00%	0 / 1
class.sqliteConnector.php	21.54%	14 / 65		20.00%	1 / 5		0.00%	0 / 1
class.virtualBoxConfiguration.php	9.41%	8 / 85		0.00%	0 / 4		0.00%	0 / 1
class.virtualBoxController.php	12.42%	19 / 153		14.29%	2 / 14		0.00%	0 / 1

Generated by PHPUnit 3.4.13 and Xdebug 2.1.0rc1 using PHP 5.2.13 at Fri Oct 15 14:06:46 CEST 2010.

Abbildung 5.8: Code-Coverage der Virtualisierungssoftware

Im Falle der Virtualisierungssoftware liegt die Code-Coverage bei etwa 35%. Warum ist dies so?
Ein grundlegendes Problem beim Testen von Software ist das Testen von privaten Funktionen. Für die Implementierung der Virtualisierungssoftware musste die Softwareumgebung verwendet werden, die in der Firma Standard ist. Nur dies gewährleistet eine optimale Kompatibiltität zu allen Windows-PCs innerhalb der Firma, da alle diese Standardumgebung verwenden. Diese Standardumgebung schließt die Verwendung von PHP 5.2.13 ein. In dieser Version von PHP gibt es Reflections, um auf andere Klassen und deren Funktionen zuzugreifen. Allerdings fehlt die Funktion *setAccessible()*, welche erst mit PHP 5.3.2 eingeführt wurde. Mit dieser Funktion kann die Sichtbarkeit von Funktionen geändert werden – z.B. von private auf public. [Vgl. Gro10] Dadurch wäre ein direkter Test privater Funktionen möglich.
Private Funktionen machen immerhin knapp 34% der Virtualisierungssoftware aus (siehe

Abbildung B.1), die somit nicht getestet werden können. Zudem beinhaltet die Virtualisierungssoftware eine große Anzahl Funktionen, die nicht oder nur sehr schwer zu testen sind, z.B. die Funktionen zum Ändern der IP-Konfiguration. Diese Funktionen sind auch sehr großteilig, da viele Prozesse untrennbar miteinander verknüpft sind und hintereinander abgearbeitet werden müssen (siehe Abschnitt 5.1). Das große Endergebnis - der Wechsel der Konfiguration, wäre somit das einzig testbare Ergebnis. Je nach Computer sollte so ein Test, der durchaus auch fehlschlagen kann, besser nicht die IP-Konfiguration verändern, um Probleme im Alltagsgeschäft zu vermeiden.
Eine Code-Coverage von knapp 35% ist somit ein guter Wert, der mit ausreichend Zeit um ein paar Prozent ausgebaut werden könnte.

5.5 Datenbanknutzung mit PDO

Während der prototypischen Implementierung der Virtualisierungssoftware trat ein Problem auf: der Wechsel des Datenbanksystems von MySQL zu SQLite. Im Normalfall hat der Programmierer dann die Aufgabe, die vorher auf MySQL getrimmten Klassen und Befehle so umzuschreiben, dass sie für SQLite funktionieren. Zwar setzen beide Datenbanken auf SQL als Abfragesprache, doch hat jedes Datenbanksystem einen eigenen Dialekt: so unterstützen manche Datenbanksysteme mehr Befehle als andere. [Vgl. Sql] Zudem ist ihre Nutzung immer verschieden. So ist zum Aufbauen einer MySQL-Verbindung ein *mysql_connect()* nötig, dem man als optionale Parameter z.B. den Servername bzw. die IP, Benutzername und Passwort übergeben kann. Für eine SQLite-Verbindung wird wiederum die Funktion *sqlite_open()* benötigt, welche als Parameter den Dateinamen der Datenbankdatei, sowie als optionale Parameter den Modus (Benutzerrechte) der Datei und eine Referenz auf eine String-Variable entgegennimmt, in die im Fehlerfall die Fehlerbeschreibung geschrieben werden soll.
Keine Funktion ist gleich benannt und auch die Parameter unterscheiden sich teils deutlich. Zumal es nicht nur SQLite und MySQL als Datenbanksysteme gibt und für diese anderen Datenbanksysteme wiederum andere Funktionen existieren. Ein Wechsel von einem Datenbanksystem zu einem anderen ist somit nur mit viel Programmieraufwand zu bewerkstelligen.
Um diesen Wartungsaufwand zu minimieren, wurde PDO entwickelt. PDO ist eine Abstraktionsebene für Datenbankzugriffe und erlaubt es, verschiedene Datenbanksysteme anzusprechen, ohne dass sich die Syntax der Funktionen oder der SQL-Statements ändert (es sei denn man verwendete SQL-Befehle, die das neue Datenbanksystem nicht beherrscht - dies gilt es zu vermeiden). Die Verwendung von PDO ist somit auch eine Möglichkeit, Funktionalität und Darstellung zu trennen, wobei die Darstellung die SQL-Query ist und die Funktionalität die Funktionen von PDO.
Die hauptsächliche Änderung beim Wechsel des Datenbanksystems ist der String, der zum Aufbau der Verbindung verwendet wird. An diesem erkennt PDO das Datenbanksystem und dessen Parameter. Um z.B. eine Verbindung zu einer MySQL-Datenbank herzustellen, wird folgender Programmcode verwendet:

```php
$host = "127.0.0.1";
$database = "test_database";
$username = "testuser";
$password = "testpassword";
try {
  $conn = new PDO("mysql:host=$host;dbname=$database", $username, $password);

} catch (PDOException $e) {
  die("Es ist ein Fehler beim Verbindungsaufbau aufgetreten.");
}
```

Listing 5.16: Verbindungsaufbau zu einer MySQL-Datenbank mit PDO

Um wiederum eine Verbindung zu einer SQLite-Datenbank aufzubauen, sieht dieser Programmcode so aus:

```php
$databaseFile = dirname(__FILE__) . "/test.db";
try {
  $conn = new PDO("sqlite:" . $databaseFile);
} catch (PDOException $e) {
  die("Es ist ein Fehler beim Verbindungsaufbau aufgetreten.");
}
```

Listing 5.17: Verbindungsaufbau zu einer SQLite-Datenbank mit PDO

Das SQLite-Beispiel fällt deshalb wesentlich kürzer aus, weil SQLite keinen Verbindungsaufbau mittels Benutzername und Passwort unterstützt (siehe Unterabschnitt 4.2.1) und die Datenbank auf dem lokalen System gespeichert ist.
Laut Kofler und Öggl [KÖ10, S. 430] sind weitere unterstützte Datenbanksysteme unter anderem Microsoft SQL Server, PostgreSQL und Firebird. Je nach Dialekt des gewählten Datenbanksystems sind leichte Anpassungen der verwendeten SQL-Statements nötig. Dies ist jedoch nicht zu vergleichen mit einem kompletten Umbau der Datenbankklasse. So ist z.B. der Programmcode zum Ausführen von SQL-Statements unter jedem Datenbanksystem *$conn->query()*, sofern vorher die Verbindung wie in Listing 5.16 oder Listing 5.17 aufgebaut worden ist.
Mit PDO wird somit der Aufwand zum Wechsel des Datenbanksystems wesentlich verringert. Zwar sind weiterhin geringe Anpassungen an den SQL-Statements notwendig, sofern SQL-Befehle verwendet worden sind, die nicht von jedem Datenbanksystem unterstützt werden, doch entfällt der Arbeitsaufwand für Anpassungen an den Datenbankklassen und den Befehlen zur Arbeit mit dem Datenbanksystem.

6 Zusammenfassung und Ausblick

6.1 Kritische Betrachtung der Ergebnisse

Das Hauptziel dieser Bachelorarbeit war, einen Prototyp einer Software zu implementieren, der zum Testen von Software für Boxen verwendet werden kann, ohne dass ein physisches Gerät zur Verfügung stehen muss. Dieses Ziel wurde erreicht. Die Virtualisierungssoftware ist relativ weit entwickelt und alle Funktionen die sichtbar sind, können auch von den Mitarbeitern verwendet werden (ausgenommen die Türsteuerung). Auch konnte auf viele Wünsche der Mitarbeiter nach Funktionen eingegangen werden, die im Programmieralltag äußerst nützlich sind, z.B. umfangreiche Debugfunktionen – Funktionen, die an einer physischen Box nicht zur Verfügung stehen. Dazu zählen z.B. das Einlegen von Dallas-Chips in geschlossene Depots, das Aufbrechen von Depots oder sogar das Senden von Kartendaten an eine andere Applikation namens „WBrowser“, welche auf echten physikalischen Boxen die Darstellung der Inhalte übernimmt. Erst durch diese Funktionen ist es für die Mitarbeiter sinnvoll, die Virtualisierungssoftware zum schnellen Testen ihrer Software einzusetzen.
Diese Konzentration auf eine gute Verwendbarkeit dessen, was an Funktionen präsentiert wird und das Erfüllen der Wünsche derer, die diese Virtualisierungssoftware am häufigsten benötigen werden, brachte jedoch Zeitprobleme mit sich. So konnten nicht alle Anforderungen erfüllt werden, die zu Anfangs aufgestellt worden sind.
Die Muss-Kriterien in der Anhang A wurden alle vier erfüllt – allerdings mit der Einschränkung, dass für alle Geräte bisher keine vorgesetzten Türen unterstützt werden. Diese fehlende Funktion ist jedoch für den Alltagsbetrieb verzichtbar.
Die Soll-Kriterien wurden nur teilweise erfüllt. So ist eine Steuerung über ein Konfigurationsskript und eine Protokollierung der Ereignisse während dieser Skriptabarbeitung aus Zeitmangel nicht implementiert worden. Die Virtualisierung der vier Gerätetypen wird zwar unterstützt, jedoch unterscheidet sich ihre virtualisierte Funktionsweise nicht. Dazu ist anzumerken, dass sich die interne Funktionsweise der Geräte auch bei den physischen Geräten gleicht und dieses Kriterium somit nicht hätte aufgenommen werden müssen. Dies war zum Verfassenszeitpunkt der Softwarespezifikation aber noch nicht bekannt.
Des Weiteren ist dem Quellcode teilweise anzusehen, dass die Techniken der Objektorientierung erst erlernt werden mussten. So entstanden Klassen wie z.B. HtmlHelper, die zwar nützlich sind, jedoch nicht ganz einfach zu verwenden sind, da z.B. bei der Verwendung der *insert()*-Funktion keine Autovervollständigung durch die IDE zur Verfügung steht, da Parameter als Array übergeben werden müssen. Mit einer besseren Objektorientierung könnte diese recht umfangreiche Klasse so gestaltet werden, dass nur die Parameter übergeben werden müssten, die auch tatsächlich benötigt werden.
Die Qualitätsanforderungen in Kapitel 7 der Anhang A wurden hingegen umfassend erfüllt.
Abschließend ist zu sagen, dass die Implementierung der Software zu geschätzten 60%

fertiggestellt ist - ein für eine prototypische Entwicklung sehr guter Wert. Die entstandene Software ist bereits ausgereift genug um produktiv eingesetzt zu werden und abgesehen von einer recht komplizierten Konfiguration über die GUI hinaus (in ini-Dateien von Anubis und Konfigurationsseiten von WinkeyNet) problemlos einsetzbar.
Mit Blick auf die Bachelorarbeit kann gesagt werden, dass sie einen guten Überblick über den Einstieg in die Programmierung mit PHP im professionellen Umfeld bietet. Erst durch das Firmenumfeld wurde sauberere Programmierung, Testen mit Unit Tests und das Erlernen von OOP und PDO thematisiert. Diese Thematisierung ist aber auch ein Nachteil: jedes dieser Themen könnte für eine eigene Bachelorarbeit herangezogen werden. Es musste deshalb eine Abwägung erfolgen bis zu welchem Detailgrad jedes dieser Themen beschrieben wird, was dazu führt, dass manche Themen nur kurz beleuchtet werden, da sie im Konzeptions- oder Implementierungsprozess nur eine untergeordnete Rolle spielten (z.B. Abschnitt 4.2). Andere Themen wie z.B. die Implementierung von Funktionen zum Ändern der IP-Konfiguration (siehe Abschnitt 5.1) fielen sehr ausladend aus, da sie viele weitere Techniken und Anwendungen beinhalteten. Diese mussten ebenfalls erläutert werden, um beim Leser das Verständnis für diese hoch entwickelten Schnittstellen zu fördern. Nicht zuletzt auch, um neugierig auf diese unter „Otto-Normal-Programmierern“ recht wenig bekannten, aber vielseitig einsetzbaren, Techniken zu machen.
Auch ist diese Bachelorarbeit eher linear entstanden, während das Softwareprojekt dahinter eher baumähnlich wuchs. Eine derartige Entwicklung ist in einer geschriebenen Arbeit nicht darstellbar, ohne unübersichtlich zu werden. So wurden Stücke des Programmcodes dutzende Male verändert, bevor sie den Stand erreichten, den man aktuell in den Quellcodes erkennen kann und die ihren Weg in diese Arbeit fanden. In mehr als 150 verschiedenen Builds wurde die Software weiterentwickelt. Ideen vom Beginn des Projekts wurden verworfen, neu implementiert und sogar die Anubis-Bibliothek, ein Kernstück der Virtualisierungssoftware, erst zur Hälfte der zur Verfügung stehenden Zeit integriert. Auch dies erfolgte erst nach zahlreichen Tests mit einer älteren, anders benannten Version dieser Bibliothek. All diese Entwicklungsschritte können in dieser Arbeit nicht berücksichtigt werden. Sie spiegelt vielmehr die Beschreibung von Techniken und Anwendungen wieder, die zum Abschluss der Programmierarbeit verwendet worden sind. Für den Leser liest sie sich deshalb wie die Dokumentation einer relativ idealen Entwicklungsarbeit. Doch auch diese Entwicklungsarbeit war von Weiterentwicklungen, verworfenem und neu überdachtem Programmcode geprägt.

6.2 Ausblick

Die mit dieser Bachelorarbeit entstandene Softwarelösung ist produktiv einsetzbar und wird auch bereits aktiv genutzt. Mitarbeiter werden die Software in Zukunft gehäuft zum Testen neuer Applikationen einsetzen und damit die Notwendigkeit von physischen Boxen für Tests abschaffen. Trotzdem sind zeitnahe Weiterentwicklungen wahrscheinlich. So fehlen noch einige wenige nützliche Debugfunktionen und die Türsteuerung muss implementiert werden. Des Weiteren ist die optische Präsentation der Software sehr

schlicht und für die Vorführung beim Kunden nur bedingt einsetzbar. Für den Einsatz bei Kundenpräsentationen stünde deshalb die Entwicklung einer optisch reizvolleren Oberfläche an.
Da die Software zudem „baumartig“ gewachsen ist, also an Teilen entwickelt wurde, die akut benötigt wurden um sie zum Laufen zu bringen oder um gravierende Fehler zu beheben, ist ihre Struktur auch nicht optimal. Auch der Quellcode selbst könnte eine Überarbeitung und eventuelle Umstrukturierung vertragen, um die zukünftige Wartbarkeit und Erweiterbarkeit des Quellcodes zu verbessern.
Folglich wären mehrere weitere Themen für Bachelor-, Diplom- und Masterarbeiten denkbar:

- „Umstrukturierung eines bestehenden Softwareprojekts zur Verbesserung der Wart- und Erweiterbarkeit unter Nutzung von Design Patterns und strikter Objektorientierung“ (Fakultät MNI)
- „Erweiterung eines bestehenden Softwareprojekts um eine automatisierbare, benutzerorientierte Skriptsteuerung mit Eventlogging“ (Fakultät MNI)
- „Entwicklung eines Designs für ein bestehendes Softwareprojekt unter Nutzung von 3D-Animation und CSS“ (Fakultät Medien)
- „Entwicklung einer Konfigurationssoftware zur Verbesserung der Benutzerfreundlichkeit einer verteilten Testsuite zum Testen von IP-basierten Sicherheitssystemen“ (Fakultät MNI)

Diese Arbeiten haben als Ziele jeweils ein großes Themengebiet, welches in dem Prototyp der Virtualisierungssoftware verbessert werden könnte.
So wird eine Arbeit mit dem Thema aus Abschnitt 6.2 den Programmcode entscheidend verbessern, denn erst während der Arbeit wurde der Umgang mit OOP gelernt und vertieft. Ein Programmierer, der bereits Erfahrungen auf diesem Gebiet hat und sich mit Design Patterns auskennt, hätte mit der Konzeptionierung der Umstrukturierung und der Umsetzung einige Zeit zu tun.
Abschnitt 6.2 bietet eine Arbeit für Programmierer mit einem durchschnittlichen Wissensstand. So müsste die Software lediglich um die Skriptsteuerung erweitert werden. Die Implementierung würde vermutlich weniger Zeit in Anspruch nehmen als der Prototyp der Virtualisierungssoftware. Somit bliebe mehr Zeit für softwaretechnische Analysen und Konzeption. Es könnte ein Schwerpunkt auf UML-Diagramme und Dokumentation gelegt werden.
Die Entwicklung eines animierten Designs mit 3D-Elementen, wie in Abschnitt 6.2 wäre ein großer Fortschritt für die Virtualisierungssoftware und würde sie auch wesentlich vorzeigbarer bei Kundengesprächen machen. Mit „animiert“ ist in dem Sinne gemeint, dass z.B. Depots sich in 3D-Darstellung öffnen oder schließen könnten oder Dallas-Chips sichtbar eingelegt werden können. Es wäre durchaus auch denkbar die 3D-Elemente durch eine AJAX-Programmierung zu ersetzen und stattdessen auf eine hochgradig intuitive Benutzerführung zu setzen, die auf Drag & Drop basiert. So könnten Benutzer Karten per Mausklick durch einen Kartenleser ziehen, Dallaschips aufnehmen und in Fächer

legen etc. Eventuell wäre es hier ratsam einen Kundenmodus mit besonders viel „Eye Candy“ einzuführen und einen Modus ohne Schnickschnack für die Programmierer.
Die unscheinbarste, aber mit Abstand schwierigste Arbeit wäre Abschnitt 6.2. Zum Konfigurieren der Virtualisierungssoftware ist es derzeit nötig, neben GUI-Einstellungen in der Software auch Anpassungen an Konfigurationsdateien von Anubis, in den Einstellungen von WinkeyNet und im WBrowser (einer Darstellungssoftware für physische Boxen) vorzunehmen. Hier können viele Flüchtigkeitsfehler entstehen die zudem Datenbankinkonsistenzen hervorrufen können, wodurch wiederum selbst bei dann korrekter Konfiguration solange Fehler entstehen, bis die Datenbanken erneuert werden. Dass dies nicht sonderlich benutzerfreundlich ist, ist klar – schließlich soll die Software auch von Personen mit weniger Computererfahrung bedient werden können. Doch dies wäre nur durch eine zusätzliche Software möglich, die die Konfiguration der einzelnen Komponenten übernimmt. Diese Komponenten können auf verschiedenen PCs liegen, zudem sollten die alten Konfigurationsdateien gesichert werden. Für WinkeyNet sind zudem Datenbankeingriffe nötig, wenn von außen Einstellungen geändert werden sollen. All diese Beschränkungen und Anforderungen machen es zu einer schwierigen Aufgabe ein Konfigurationssystem für diese Komponenten zu entwickeln. Vor dieser Arbeit sollte aber das Praktikum in dieser Firma absolviert werden um sich mit den verschiedenen Komponenten auszukennen, bevor dafür Konfigurationstools entwickelt werden sollen.
Es ist festzustellen, dass genügend Arbeit nach dieser Bachelorarbeit übrig geblieben ist. Es wurden sogar neue Möglichkeiten für weitere Bachelor-, Diplom- und Masterarbeiten geschaffen, was aufzeigt, wie komplex und vielschichtig das Thema dieser Bachelorarbeit war.

Anhang A: Softwarespezifikation

Gesellschaft für Elektronik, Elektrotechnik, Mechanik und Systeme mbH

Softwarespezifikation

Virtualizing

Version 0.1

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Anforderungen

1.1 Muss-Kriterien

- Grafische Darstellung der virtuellen Box
- Einstellmöglichkeiten über GUI (Firmware, BOX ID, Boxaufbau)
- Virtualisierung des Controllers von mindestens 2 KEMAS-Produkten (Keybox, Unibox)´
- Lauffähig auf einem PC mit Windows Betriebssystem (ab XP)

1.2 Soll-Kriterien

- Einstellmöglichkeiten über Script
- Protokollierung der Ereignisse während einer Scriptabarbeitung
- Virtualisierung des Controllers von 4 KEMAS-Produkten (Keybox, Unibox, Storebox, Fachanlage)

1.3 Kann-Kriterien

- Einhaltung der KEMAS-CI
- Optisch ansprechende Darstellung der Box

1.4 Abgrenzungskriterien

- Keine Darstellung der Box in 3D
- Virtualisierung eines SBC (inkl. Benutzerrechten) ist nicht möglich
- Die Software ist nicht lauffähig auf einem PC ohne Windows Betriebssystem

2 Produkteinsatz

2.1 Anwendungsbereiche

Das Produkt wird zur Verwendung in zwei Anwendungsbereichen entwickelt:

1. zur *internen Verwendung* als Hilfsmittel zum schnelleren und unkomplizierteren Entwickeln und Debuggen von Programmen, die normalerweise die Anwesenheit eines physischen Geräts erfordern (welches erst umständlich händisch bedient werden müsste).
2. zur *Vorführung beim Kunden* um die Mitnahme eines physischen Geräts überflüssig zu machen.

2.2 Zielgruppen

In jedem der zwei Hauptanwendungsbereiche wird die Software von mit den physischen Produkten vertrauten Anwendern und Programmierern bedient. Dass die Software von Außenstehenden benutzt wird, ist nahezu ausgeschlossen.

2.3 Betriebsbedingungen

Die Software kann auf einem herkömmlichen x86-PC-System verwendet werden. Je nach Einsatzzweck als Hilfsmittel beim Debuggen oder zur Vorführung beim Kunden ist keine ständige Überwachung notwendig (automatisierter Scriptbetrieb) oder die GUI-gestützte Bedienung und Überwachung erforderlich.

3 Produktübersicht

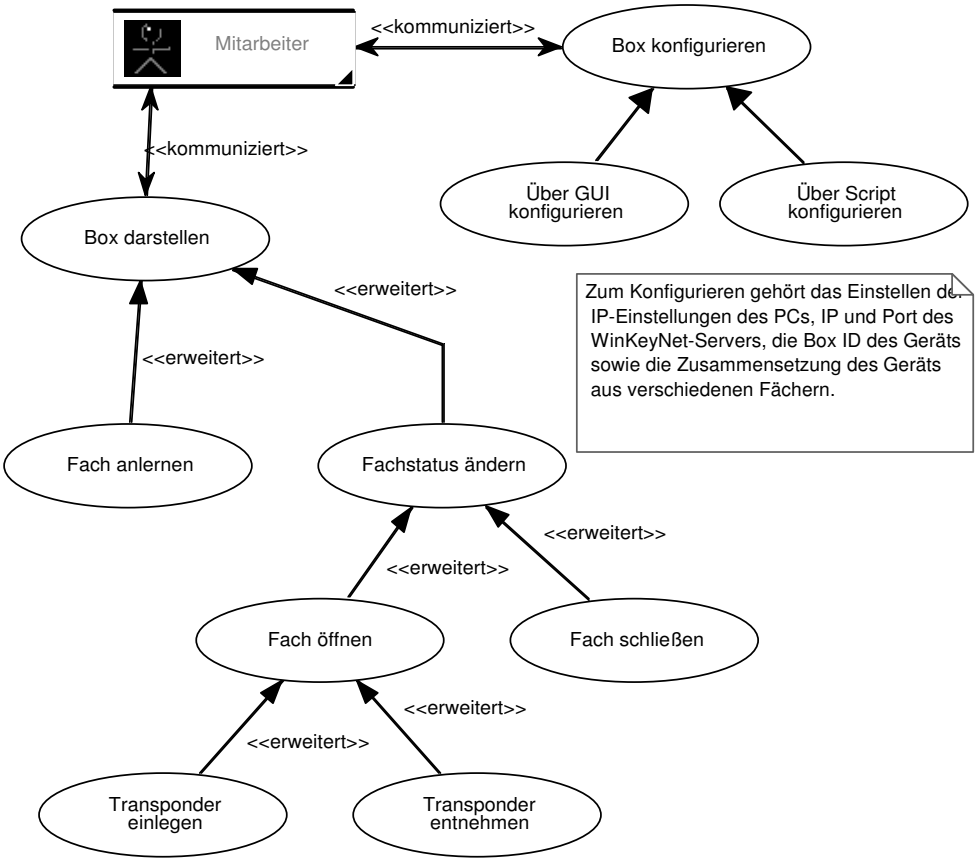


Abbildung 1: Use-Case-Diagramm

Das Produkt ist ein auf einem Webserver (z.B. Apache) lauffähiges PHP-Script, welches es ermöglicht Hardwaregeräte der Firma KEMAS zu simulieren. Es verwendet dafür die VirtualBox als Zwischenschicht zwischen Virtualizing und Anubis/WinkeyNet.

3.1 Erste Version

In einer ersten Version soll das Frontend, das der Endbenutzer bedienen kann, voll funktionsfähig vorhanden sein. So kann der Benutzer sich eine Box aus verschiedenen KEMAS-Geräten zusammenstellen und sich den Status der einzelnen Fächer anzeigen lassen. Eine erste Kommunikation mit der VirtualBox (z.B. Öffnen und Schließen eines Fachs) soll möglich sein.

3.2 Zweite Version

In der zweiten Version kann das Script Byteströme verarbeiten, die der Steuerung der in der (Virtual-)Box enthaltenen Fächer etc. dienen. Es kann so auch dem Anubis- bzw. WinkeyNet-System Statusmeldungen machen. Weiterhin sind in dieser Anzeige Eingaben wie eingelegte Schlüssel, eingelesene Karten etc. möglich.

3.3 Dritte Version

Die dritte Version soll grafisch verbessert sein und somit auch bei Vorführungen beim Kunden einsetzbar sein. Es wird „Eye Candy“ wie der KEMAS-Header und -Footer, sowie weitere Bilder und Grafiken eingefügt.

4 Produktfunktionen

Es ist nicht das Ziel dieses Pflichtenheftes, seitenweise Funktionsbeschreibungen zu beherbergen. Um die Beschreibung der Funktionen kurz zu halten, werden *User Stories* verwendet. Diese beschreiben die gewünschte Funktionalität in einfachen Sätzen.

Konfigurieren (von Hand)	Als Mitarbeiter kann ich durch Eingaben in einer grafischen Oberfläche alle Einstellungen vornehmen, damit das Script ordnungsgemäß funktioniert.
Konfigurieren (per Script)	Als Mitarbeiter kann ich ein Konfigurationsscript laden, damit ich keine Einstellungen über die Programmoberfläche mehr vornehmen muss.
IP-Einstellungen ändern	Als Mitarbeiter kann ich über die Scripteinstellungen eine Änderung der IP-Einstellungen – dazu gehören IP-Adresse, Subnetzmaske und Standardgateway – bewirken.
Apache-Port ändern	Als Mitarbeiter kann ich den Port, auf dem sämtliche Webseiten auf meinem PC erreichbar sind (also einschließlich Virtualizing und VirtualBox), durch Scripteinstellungen ändern.
Boxspezifische Einstellungen vornehmen	Als Mitarbeiter kann ich über Scripteinstellungen festlegen, aus welchen Teilgeräten meine Box besteht und wie groß diese Teilgeräte in ihrer Höhe und Breite sind.
Box darstellen	Als Mitarbeiter kann ich mir nach der Durchführung der Konfiguration die Box anzeigen lassen. In der Darstellung kann ich Fächer öffnen und schließen, Transponder einlegen, Karten einlesen und den Fachstatus verändern.
Fach anlernen	Als Mitarbeiter kann ich in der Darstellung einem Fach mitteilen, welche Transpondernummer für es die richtige ist.
Fach öffnen/schließen	Als Mitarbeiter kann ich in der Darstellung ein Fach öffnen und schließen.
Fachstatus ändern	Als Mitarbeiter kann ich in der Darstellung den Status eines Faches verändern.
Tür öffnen/schließen	Als Mitarbeiter kann ich in der Darstellung eine vorgesetzte Tür vor einem Teilgerät öffnen oder schließen.
Transponder in/aus Fach legen/entnehmen	Als Mitarbeiter kann ich in der Darstellung einen Transponder aus einem geöffneten Fach entnehmen, bzw. einen in ein geöffnetes Fach hineinlegen.
Fachstatus erkennen	Als Mitarbeiter kann ich in der Darstellung den Fachstatus an einer stilisierten LED an jedem Fach erkennen.

Abbildung 2: Beschreibung der Produktfunktionen als User Stories

Welche Klassen wodurch instanziert werden, wird im Klassendiagramm dargestellt. Dieses ist in Abbildung 3 zu sehen.

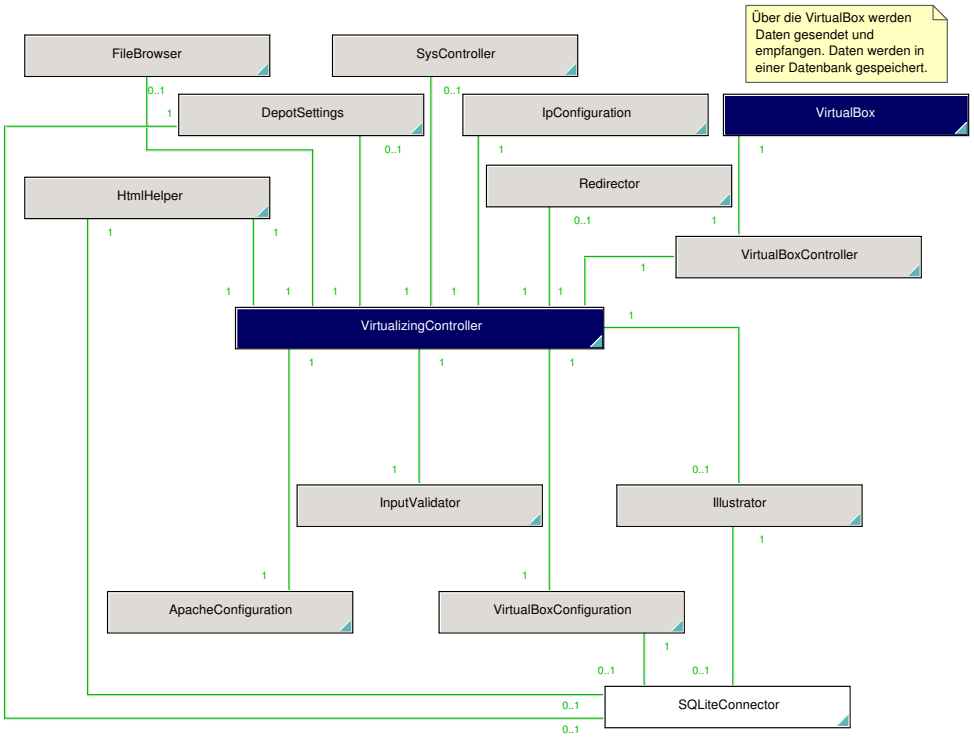


Abbildung 3: Klassendiagramm

4.1 Aktivitätsdiagramm „Box konfigurieren“

Um den Datenfluss innerhalb des Programms zu demonstrieren und einen groben Einblick in die Funktionsweise der Virtualizing-Software zu erhalten, zeigen die nachfolgenden zwei Seiten das Aktivitätsdiagramm für den Use Case „Box konfigurieren“.
Dieser Use Case ist einer der komplexesten des ganzen Programms und zeigt das Zusammenspiel zwischen Klassen, Daten und dem Benutzer. Es wurde absichtlich auf eine Unterteilung in kleinere Teile des Use Cases verzichtet, da im Realeinsatz das hier gezeigte Stück Code unmittelbar abläuft.

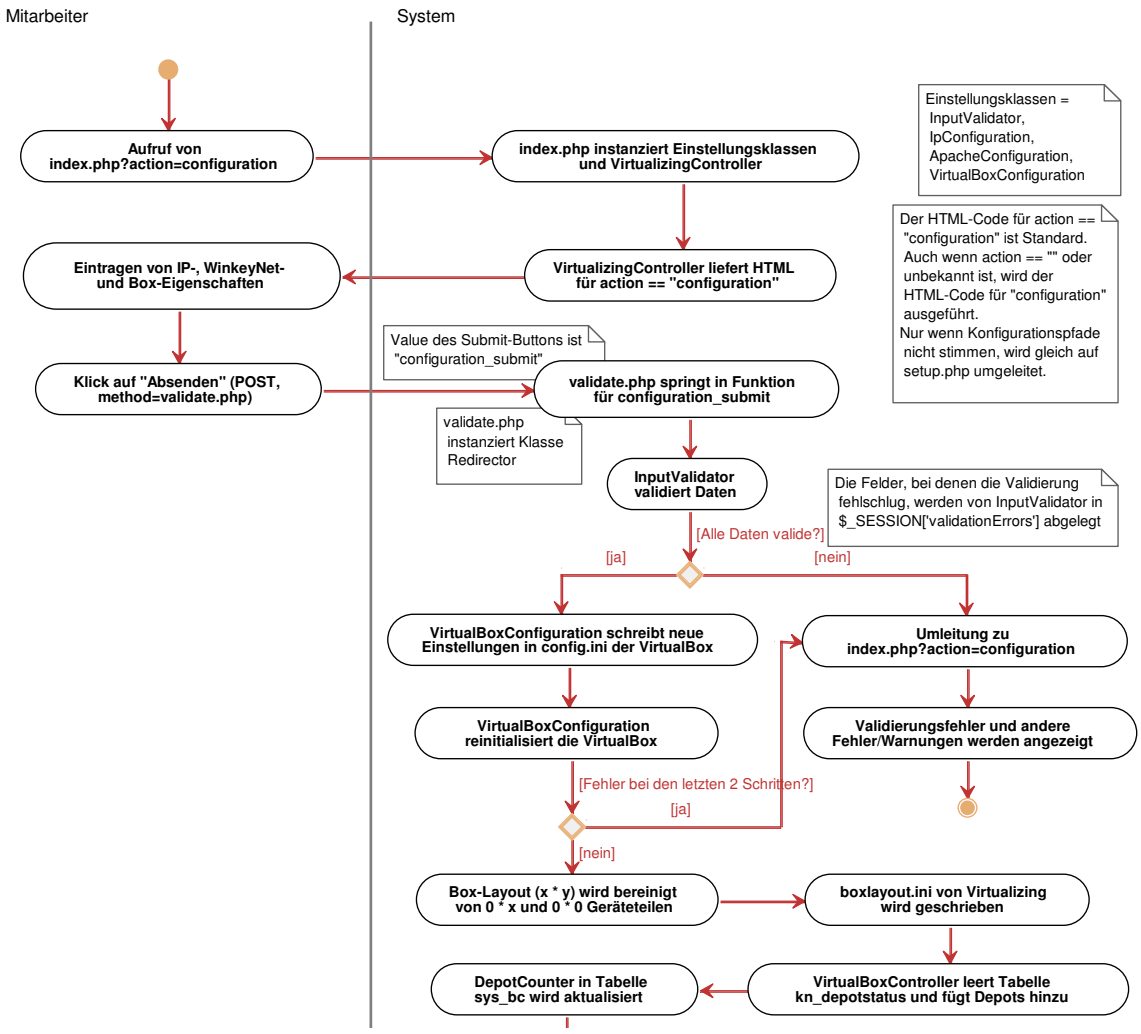


Abbildung 4: Aktivitätsdiagramm „Box konfigurieren“, Teil 1

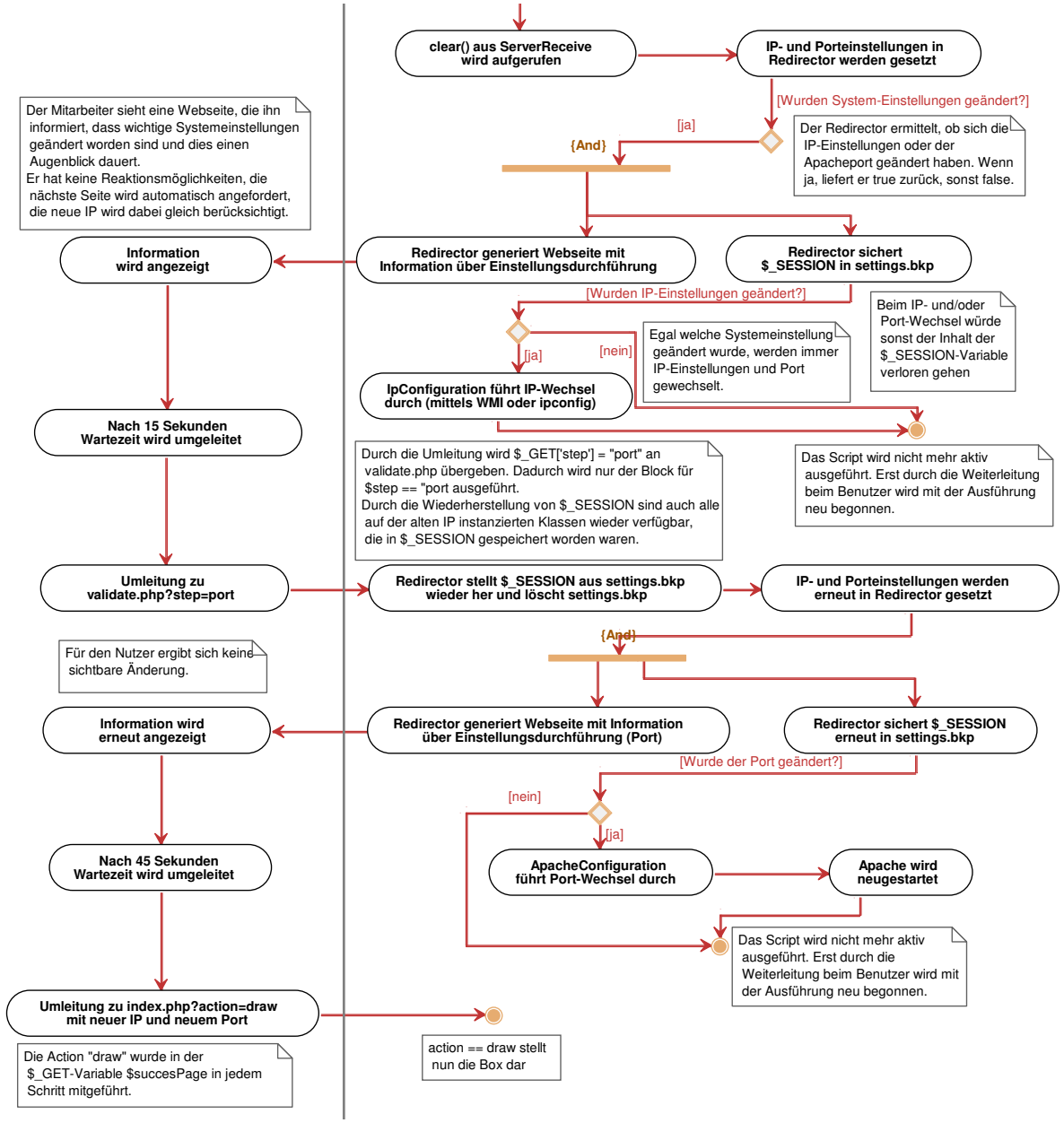


Abbildung 5: Aktivitätsdiagramm „Box konfigurieren“, Teil 2

5 Produktdaten

In Abbildung 6 werden die Daten dargestellt, die die Virtualisierungssoftware verwendet.

/D10/	**Aufbau/Layout der Box**	Höhe und Breite der einzelnen Komponenten
/D20/	**Eingelegte Schlüssel und Karten**	Transpondernummer, Fachnummer, Zeitpunkt der letzten Änderung
/D30/	**Depotstatus**	leer, belegt, blockiert, defekt

Abbildung 6: Produktdaten

6 Produktleistungen

In Abbildung 7 ist dargestellt, welche Leistungen das Produkt erbringen muss.

/L10/	**Benutzerfreundlichkeit**	Bei falschen oder nicht sinnvollen Eingaben und anderen Problemen soll der Benutzer aussagekräftige Fehlermeldungen und Warnungen erhalten.
/L20/	**Toleranz**	Bei falschen Eingaben wird der Benutzer auf die Seite zurückgeleitet, auf der er die falschen Eingaben vorgenommen hat. Die von ihm angegebenen Daten werden angezeigt und falsche Werte hervorgehoben.
/L30/	**IP-Konfiguration**	Das Einstellen von IP und (Apache-)Port des PCs, auf dem das Script läuft, soll innerhalb von 70 Sekunden abgeschlossen sein.

Abbildung 7: Produktleistungen

7 Qualitätsanforderungen

Abbildung 8 zeigt eine Übersichtstabelle über die Gewichtung der einzelnen Schwerpunkte bei der Softwareentwicklung.

Produktqualität	*sehr gut*	*gut*	*normal*	*nicht relevant*
Funktionalität	x			
Zuverlässigkeit		x		
Benutzbarkeit		x		
Effizienz			x	
Änderbarkeit			x	
Übertragbarkeit				x

Abbildung 8: Qualitätsanforderungen in der Übersicht

In Abbildung 9 werden die einzelnen Gewichtungen genau erläutert.

Funktionalität	Die Software soll ein echtes Hardwaregerät mit allen Funktionen virtualisieren, die man zum Testen in der täglichen Entwicklung und zur Vorführung beim Kunden benötigt. Aufgrund der Bewertung mit *sehr gut*, müssen diese Funktionen auf Protokollebene so funktionieren, wie beim Echthardwaregerät auch.
Zuverlässigkeit	Vorausgesetzt das Script wird korrekt bedient und konfiguriert, so soll es zuverlässig arbeiten. Da dieser Punkt nur mit *gut* gewichtet wurde, wird bei Zwischenfällen wie Ruhezustand oder Standby des Computers ein Benutzereingriff notwendig sein, um die Software wieder anzustoßen. Das Löschen von benötigten Dateien und Daten in Datenbanken während der Laufzeit wird den Scriptablauf negativ beeinflussen.
Benutzbarkeit	Die Software soll für jeden Mitarbeiter der Firma intuitiv zu benutzen sein. Da es sich um eine Webanwendung handelt, werden lediglich Kenntnisse zur Benutzung eines Internetbrowsers benötigt. Kenntnisse über zu virtualisierende Hardware und mindestens geringfügige Computerkenntnisse sind Voraussetzung. Da dieser Punkt mit *gut* gewichtet wurde, wird auf eine harmonische GUI-Gestaltung und Menüführung geachtet. Es wird aber kein zusätzlicher Aufwand im Sinne von Benutzerbarkeitsstudien o.ä. betrieben.
Effizienz	Da die Software als Webanwendung läuft, halten sich Ladezeiten in Grenzen. Da die Software zudem praktisch immer lokal auf dem Rechner eingesetzt wird, auf dem Software gegen das virtualisierte Gerät getestet werden soll, verringern sich die Latenzen gegenüber der Datenübertragung übers Netzwerk nochmals deutlich. Reaktionszeiten von unter 3 Sekunden sollten jedoch in jedem Fall die Regel sein. Abhängig von der Polling-Frequenz bei der Darstellung können sich aus der tatsächlichen Antwortzeit und der Zeit bis zum nächsten Polling im Worst Case Wartezeiten zwischen 8 und 10 Sekunden ergeben. Je höher die Polling-Frequenz, desto mehr Prozessorlast wird erzeugt und desto mehr RAM benötigt, da Datenbankzugriffe und PHP-Compilerdurchläufe proportional steigen. Man kann somit zwischen kürzeren Reaktionszeiten und geringerer Prozessorlast wählen.
Änderbarkeit	Die Software wird in der Programmiersprache PHP geschrieben und arbeitet größtenteils objektorientiert. PHP ist eine sehr weit verbreitete Scriptsprache zur Entwicklung von Webanwendungen. Die Bearbeitung kann mit einem simplen Editor (z.B. Notepad) bereits durchgeführt werden. Höher entwickelte Editoren wie z.B. Zend Studio oder PHPDesigner sind aber für eine effizientere Bearbeitung empfehlenswert. Kommentare im Quelltext, die reichlich vorhanden sind, verbessern die Änderbarkeit zusätzlich.
Übertragbarkeit	Da die Software lediglich auf Windows-PCs (ab Windows XP) laufen soll, ist eine Übertragbarkeit nicht relevant. Die einzige Übertragung die stattfinden könnte, wäre von einem Windows-PC auf einen anderen (durch kopieren). Danach müssen eventuell Einstellungen neu vorgenommen werden. Eine Setupfunktion hilft einem bei der schnellen Einrichtung.

Abbildung 9: Erläuterung der Gewichtung der Qualitätsanforderungen

8 Technische Produktumgebung

- Software
  - Server (mit Virtual Box):
    - Webbrowser mit CSS-Unterstützung (bei Standaloneverwendung)
    - Betriebssystem mit Apache-Webserver und PHP
    - Datenbanksystem (SQLite)
  - Client:
    - Webbrowser mit CSS- und Javascript-Unterstützung
- Hardware
  - x86-PC
  - Netzwerkkarte (bei Verwendung von Virtual Box, Client und Anubis/WinkeyNet übers Netzwerk)
- Orgware
  - Netzwerkverbindung (LAN) (bei Verwendung von Virtual Box, Client und Anubis/WinkeyNet übers Netzwerk)

Anhang B: Zusätzliche Bilder

```
phploc 1.5.1 by Sebastian Bergmann.

Directories:                                        3
Files:                                             38

Lines of Code (LOC):                             5158
  Cyclomatic Complexity / Lines of Code:         0.11
Comment Lines of Code (CLOC):                     972
Non-Comment Lines of Code (NCLOC):               4186

Namespaces:                                        0
Interfaces:                                        0
Classes:                                          28
  Abstract:                                        0 (0.00%)
  Concrete:                                       28 (100.00%)
  Average Class Length (NCLOC):                  126
Methods:                                         145
  Scope:
    Non-Static:                                  139 (95.86%)
    Static:                                        6 (4.14%)
  Visibility:
    Public:                                       96 (66.21%)
    Non-Public:                                   49 (33.79%)
  Average Method Length (NCLOC):                  24
  Cyclomatic Complexity / Number of Methods:     3.36

Anonymous Functions:                               0
Functions:                                        12

Constants:                                        11
  Global constants:                               11
  Class constants:                                 0
```

Abbildung B.1: Codeauswertung mit PHPLoc

Abbildung B.2: Der erste Schritt den der Benutzer sieht, zum Einstellen der Pfade

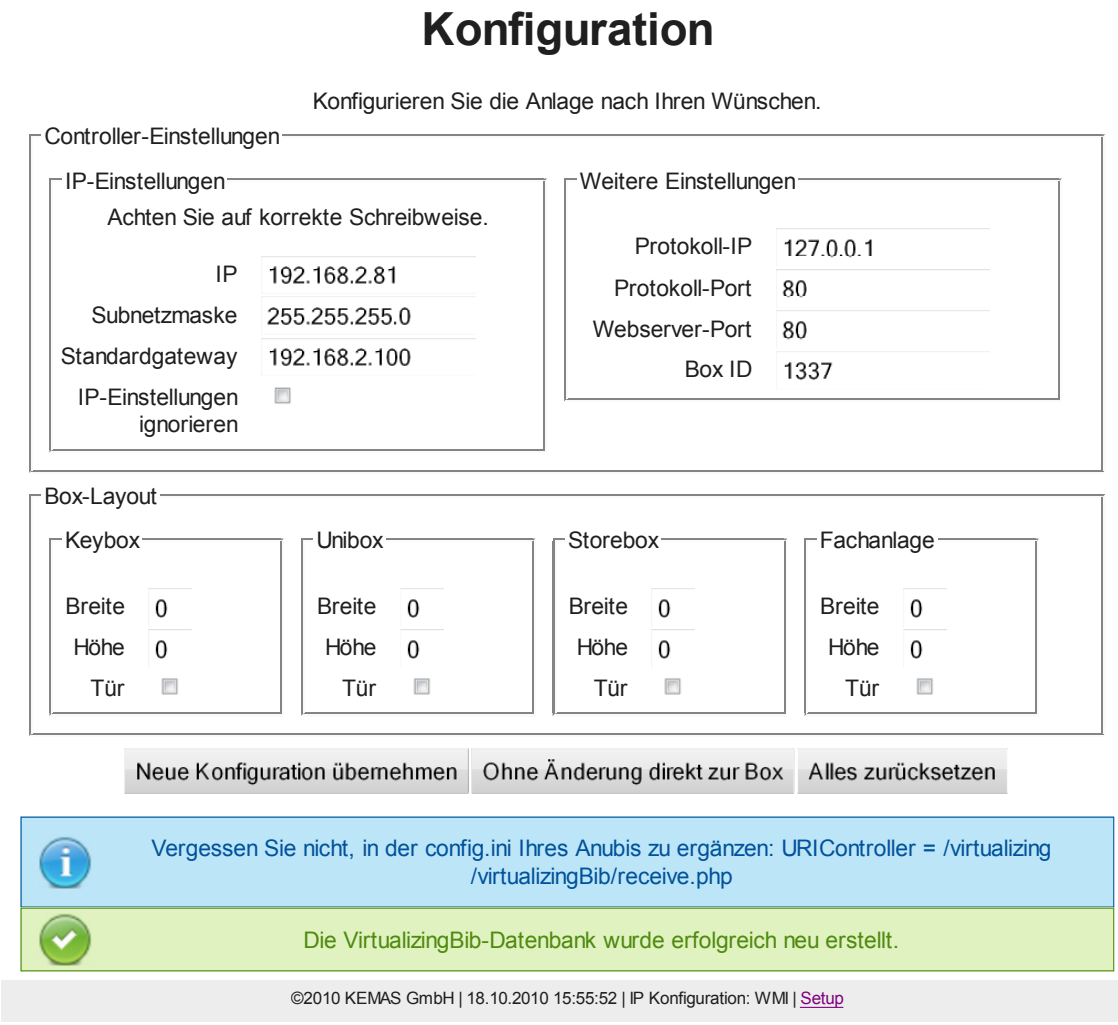


Abbildung B.3: Der zweite Schritt, die Konfiguration der virtuellen Box

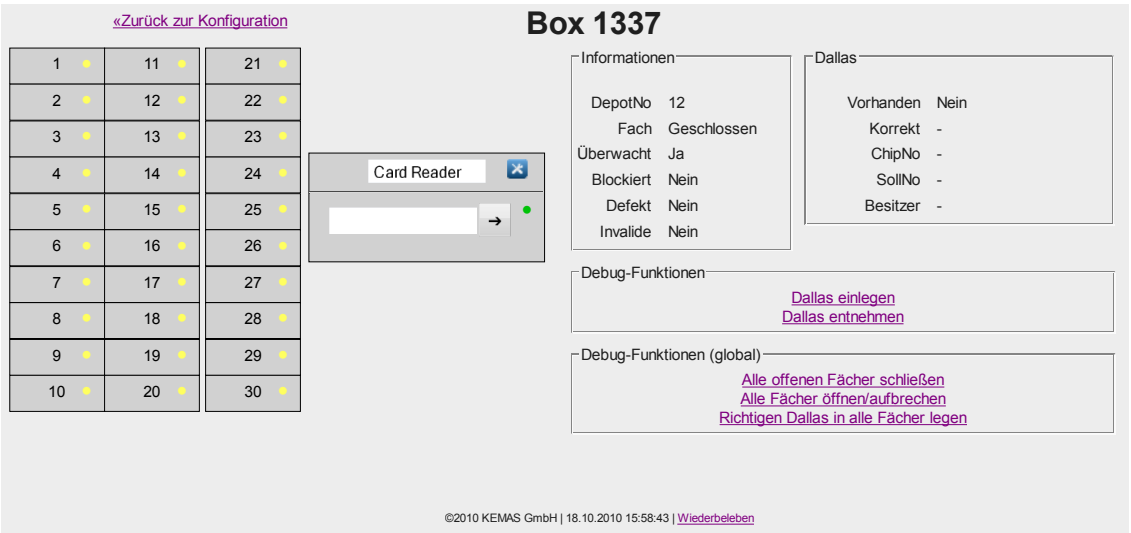


Abbildung B.4: Der dritte Schritt, die Darstellungsseite nach einem Klick auf Depot 12

Literaturverzeichnis

[Ach+10] Mehdi Achour u. a. *PHP Manual*. Verfügbar am 08.10.2010. 2010. URL: `http://de3.php.net/get/php_manual_de.html.gz/from/a/mirror`.

[Bal09] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. 3. Aufl. Springer-Verlag GmbH, 2009.

[Fla05] David Flanagan. *Java in a nutshell: deutsche Ausgabe für Java 1.4*. 2., korr. Nachdruck. O'Reilly Verlag GmbH & Co. KG, 2005.

[Fla07] David Flanagan. *Javascript: Das umfassende Referenzwerk*. 5. Aufl. O'Reilly Media, 2007.

[GBR05] Andi Gutmans, Stig Saether Bakken und Derick Rethans. *PHP 5 aus erster Hand*. Addison-Wesley Verlag, 2005.

[Gil08] W. Jason Gilmore. *Beginning PHP and MySQL: from novice to professional*. Apress Verlag, 2008.

[Gro10] The PHP Group. *ReflectionMethod::setAccessible*. Verfügbar am 18.10.2010. 2010. URL: `http://php.net/manual/de/reflectionmethod.setaccessible.php`.

[Gün04] Ulrich Günther. *PHP 5 - Ein praktischer Einstieg*. 2. Aufl. O'Reilly Media, 2004.

[Hai06] Paul Haine. *HTML Mastery: Semantics, Standards, and Styling*. Apress Verlag, 2006.

[HB05] Craig Hunt und Roberta Bragg. *Windows Server 2003 Network Administration*. O'Reilly Media, 2005.

[HHI07] Hartmut Helmke, Frank Höppner und Rolf Isernhagen. *Einführung in die Softwareentwicklung: Vom Programmieren zur erfolgreichen Softwareprojektarbeit*. Carl Hanser Verlag München Wien, 2007.

[Hud06] Paul Hudson. *PHP in a nutshell*. O'Reilly Media, 2006.

[Häß03] Ulrike Häßler. *Cascading Stylesheets: Stil mit 'Stil'*. Springer-Verlag GmbH, 2003.

[Inf02] Information Gatekeepers Inc. *Scalable Ethernet Networks for LAN, MAN & WAN*. 2002.

[Kan07] Matthias Kannengiesser. *Objektorientierte Programmierung mit PHP 5*. Franzis Verlag GmbH, 2007.

[Ker06] Christian Kern. *Anwendung von RFID-Systemen*. 2., überarbeitete Aufl. Springer-Verlag GmbH, 2006.

[Kra04] Jörg Krause. *Programmieren lernen in PHP 5*. Carl Hanser Verlag München Wien, 2004.

[KS09] Guido Krüger und Thomas Stark. *Handbuch der Java-Programmierung*. 5. Aufl. Addison-Wesley Verlag, 2009.

[KÖ10] Michael Kofler und Bernd Öggl. *PHP 5.3 & MySQL 5.4: Programmierung, Administration, Praxisprojekte*. Addison-Wesley Verlag, 2010.

[Lar05] Dirk Larisch. *Netzwerktechnik*. 2., überarbeitete Aufl. verlag moderne industrie Buch AG & Co. KG, Landsberg, 2005.

[Ler+07] Rasmus Lerdorf u. a. *Programmieren mit PHP*. 2. Aufl. O'Reilly Media, 2007.

[LM01] Matthew Lavy und Ashley Meggitt. *Windows Management Instrumentation (WMI)*. New Riders Publishing, 2001.

[LN08] Hoa Loranger und Jakob Nielsen. *Web Usability*. Addison-Wesley Verlag, 2008.

[Mau09] Florence Maurice. *PHP 5.3 & MySQL 5.1: Der Einstieg in die Programmierung dynamischer Websites*. Addison-Wesley Verlag, 2009.

[MK06] Chuck Musciano und Bill Kennedy. *HTML & XHTML: The Definitive Guide*. O'Reilly Media, 2006.

[Ms0] *Geheimnisse von Windows Management Instrumentation - Problembehandlung und Tipps*. Verfügbar am 11.10.2010. Microsoft Corporation. 2004. URL: `http://www.microsoft.com/germany/technet/datenbank/articles/600682.mspx`.

[MS04] Christoph Meinel und Harald Sack. *Www: Kommunikation, Internetworking, Web-technologien*. Springer-Verlag GmbH, 2004.

[Ms1] *WMI Return Codes*. Verfügbar am 13.10.2010. Microsoft Corporation. 2010. URL: `http://msdn.microsoft.com/en-us/library/aa394574%28VS.85%29.aspx`.

[Mün08] Stefan Münz. *Webseiten professionell erstellen: Programmierung, Design und Administration von Webseiten*. 3., überarbeitete und erweiterte Aufl. Addison-Wesley Verlag, 2008.

[Nas07] Marcus Nasarek. *Windows Vista Security: praxisorientierte Sicherheit für Profis*. O'Reilly Media, 2007.

[PM05] Christine Peyton und André Möller. *PHP 5.1 und MySQL 4.1*. Markt + Technik Verlag, 2005.

[Roe09] Michael Roeschter. *Java Service Launcher (for Windows) JSL 0.99l*. Verfügbar am 17.09.2010. 2009. URL: `http://jslwin.sourceforge.net/`.

[Sch10] Holger Schwichtenberg. *Windows Scripting: Automatisierte Systemadministration mit dem Windows Script Host 5.8 und der Windows PowerShell 2.0*. 6., aktualisierte Aufl. Addison-Wesley Verlag, 2010.

[SKTR05] Holger Schwichtenberg, Stephanie Knecht-Thurmann und Manuela Reiss. *Windows XP Professional: Das Profi-Handbuch für den Unternehmenseinsatz*. Addison-Wesley Verlag, 2005.

[Spu00] Charles E. Spurgeon. *Ethernet: The Definitive Guide*. First Edition. O'Reilly Media, 2000.

[Sql] *SQL Features That SQLite Does Not Implement*. Verfügbar am 17.10.2010. 2010. URL: `http://www.sqlite.org/omitted.html`.

[SRJ08] Holger Schwichtenberg, Manuela Reiss und Thomas Joos. *Windows Vista Business: Das Profi-Handbuch für den Unternehmenseinsatz*. Addison-Wesley Verlag, 2008.

[Sta04] Dieter Staas. *PHP 5 Espresso!* Franzis Verlag GmbH, 2004.

[SW08] William Steinmetz und Brian Ward. *Wicked Cool PHP: Real-World Scripts That Solve Difficult Problems*. No Starch Press, 2008.

[TT10] Gerrit Tamm und Christoph Tribowski. *RFID*. Springer-Verlag GmbH, 2010.

[TW09] Laura Thomson und Luke Welling. *PHP 5.3 & MySQL 5.1: Dynamische Anwendungen von Einstieg bis E-Commerce*. Markt + Technik Verlag, 2009.

[Wen07] Christian Wenz. *Programmieren mit ASP.NET AJAX*. O'Reilly Media, 2007.

[Wol04] Sebastian Wolfgarten. *Apache Webserver 2*. Addison-Wesley Verlag, 2004.

[Zan10] Matt Zandstra. *PHP Objects, Patterns and Practice*. Apress Verlag, 2010.

Glossar

AJAX Asynchronous JavaScript and XML, eine Technik zur asynchronen Datenübertragung um Daten zwischen Server und Browser zu übertragen, obwohl die Webseite bereits geladen ist; es können auch Teile der Webseite aktualisiert werden, ohne dass die gesamte Webseite neu geladen werden muss

ATM Asynchronous Transfer Mode, ist ein Netzwerkstandard der - im Gegensatz zu Ethernet - eine feste Paketgröße von 53 Byte verwendet und zu Ethernet vollkommen inkompatibel ist

Auto-ID Automatische Identifikation

Backend Vom Endbenutzer weiter entfernter Soft- oder Hardwareteil, meist mit administrativer Funktion

Box Andere Bezeichnung für ein Gerät

Build Zwischenstand des Programmcodes einer Software

Bytecode Ein betriebssystemunabhängiger Zwischencode, der mithilfe einer virtuellen Maschine ausgeführt werden kann

C165 16-Bit Microcontroller, hergestellt von Infineon

CLI Command Line Interface, im Zusammenhang mit PHP sind damit PHP-Skripte gemeint, die in der Kommandozeile abgearbeitet werden

Code-Coverage Abdeckung des Programmcodes mit Tests in %

COM Component Object Model, eine objektorientierte Kommunikationsschicht, die es einem erlaubt, Programmcode aus anderen Programmen und DLLs zu verwenden

Content Der eigentliche Inhalt, bei HTML die auf dem Bildschirm dargestellte Information

Controller Teil des Model/Control/View-Softwaremodells, verarbeitet Daten gemäß Benutzeraktionen

CRUD Abkürzung für Create, Read, Update und Delete - die Hauptbefehle bei SQL

CSS Cascading Style Sheets, eine Formatierungssprache, um die Darstellung von Webseiten zu konfigurieren

Dallas Synonym für RFID-Transponder, da der Hersteller in diesem Fall Dallas ist

DBMS Datenbankmanagementsystem

Design Patterns Entwurfsmuster, ein bewährter Lösungsvorschlag für bekannte Probleme in der Softwaretechnik und -entwicklung

Duale Lizenz Mehrfachlizenzierung, bei der z.B. eine Open-Source-Lizenz und eine proprietäre Lizenz kombiniert werden

Ethernet Technik für ein kabelgebundenes Datennetz mit Geschwindigkeiten von 10 MBit/s - 10 GBit/s

Executable Ausführbare Datei

Eye Candy Anderes Wort für einen Augenschmaus, Animationen und Farben, böswillig auch als „Blendwerk“ bezeichnet

Freie Software Freie Software darf für jeden Zweck verwendet und bearbeitet werden, einschließlich kommerzieller Nutzung

GPL GNU General Public License, Lizenzmodell für die Lizenzierung Freier Software

GUI Graphical User Interface

HTML Hyper Text Markup Language, Auszeichnungssprache zur Inhaltsstrukturierung, z.B. von Texten und Bildern

IDE Integrated Development Environment, eine Software zum Schreiben von Programmen

Instanz Exemplar einer Klasse

JDBC Java Database Connectivity, eine Schnittstelle in Form eines Treibers zwischen der Anwendung und der SQL-Datenbank

JRE Java Runtime Environment, Software, die die Übersetzung des Bytecodes in Maschinenbefehle durchführt

JSL Java Service Launcher, Software, die ein Javaprogramm als Systemdienst kapseln kann

JVM Java Virtual Machine, virtuelle Maschine, die Bytecode ausführt

Klasse „Bauplan“ für einen bestimmten Typ eines Objekts

Magische Konstante Eine von der Programmiersprache bereitgestellte Konstante, die automatisch mit einem bestimmten Inhalt gefüllt wird

Meta-Element Ein leeres Element in HTML-Dokumenten um Metadaten, z.B. den Autor oder die Zielgruppe anzugeben

Moniker Ein anderes Wort für „Pseudonym“, in diesem Zusammenhang ein String zur Objektidentifizierung

MVC-Modell Model-View-Controller-Modell, ein Modell zur Trennung von Datenbank (Model), Darstellung (View) und Controller (Steuerung)

MySQL DBMS unter GPL, im Internet extrem häufig genutzt

Netsh Network Shell, ist ein Kommandozeilenprogramm zur Überwachung und Konfiguration von Netzwerkinterfaces

Objekt Instanz einer Klasse mit eigenen Eigenschaften und Methoden

OCR Optical Character Recognition

OOP Objektorientierte Programmierung

Open-Source Sammlung von Lizenzen und Software für quelloffene Software, um die Weiterentwicklung zu fördern

Output Hier: Bildschirm- oder Logdateiausgabe

PDO PHP Data Objects, Abstraktionsebene für den Zugriff auf verschiedene Datenbanksysteme mit gleichbleibender Syntax

PEAR PHP Extension and Application Repository, eine Bibliothek von nützlichen Funktionen, Modulen und Erweiterungen für PHP

Plugin Meist kleine Software, die ein anderes Programm um eine oder mehrere Funktionen erweitert, englisch „einklinken“ = plug in

PMD Es gibt keine Ausschrift für diese „Abkürzung“, es handelt sich um ein Projekt zur Verminderung der Komplexität von Funktionen und Klassen

Polling Eine Technik, bei der Informationen in einem festgelegten Takt regelmäßig abgefragt werden, egal ob Änderungen der Informationen eintraten oder nicht

Public Domain Frei von allen Rechten, unterliegt auch nicht dem Urheberrecht, es werden uneingeschränkte Nutzungsrechte eingeräumt

Referenz Ein Verweis auf ein Objekt

Relationale Datenbank Sammlung von Tabellen, welche aus Spalten und Zeilen bestehen

Resultset Sammlung von Datensätzen aus einer Datenbank inklusive Metadaten wie Spaltennamen etc.

Retina Netzhaut des Auges

RFID Radio Frequency Identification

SBC Single Board Computer, ein kleiner PC auf einer einzigen Platine

Service Dienst, eine Software, die ständig im Hintergrund läuft und meist mit einer Anwendung im Vordergrund kommuniziert

SNMP Simple Network Management Protocol, ein Protokoll zur zentralen Überwachung und Steuerung von Netzwerkkomponenten inkl. Peripherie, z.B. Druckern

Spaghetticode Schlecht strukturierter Programmcode, der für andere Programmierer schwer verständlich ist

SQL Structured Query Language, Datenbanksprache für relationale Datenbanken

SQL-Query Datenbankabfrage in einer SQL-Datenbank

SQLite Open-Source DBMS das je Datenbank eine Textdatei verwendet

Superglobales Array Ein globales Array in PHP, das in sämtlichen Klassen und Skriptteilen verfügbar ist, ohne den Einsatz des „global“ Schlüsselworts

TCP Transport Control Protocol, ist ein verbindungsbasierendes Internetprotokoll mit Fehlerkorrektur

Transponder Funkgerät, das automatisch auf eingehende Signale antwortet, Wortmischung aus Transmitter (= Übertrager) und Responder (= Beantworter)

Try-Catch-Umgebung Der Bereich innerhalb der geschweiften Klammern nach dem Schlüsselwort try, sowie dem zugehörigen Schlüsselwort catch

UDP User Datagram Protocol, ist ein verbindungsloses Internetprotokoll ohne Fehlerkorrektur - diese muss die Anwendung selbst übernehmen

Unit Test Software kann in kleinere Teile, sogenannte Units zerlegt werden, diese können dann automatisiert mit Ausgangsdaten und einem bekannten korrekten Ergebnis getestet werden

User Story Kurze Funktionsbeschreibung in Form einer Mini-Geschichte, erzählt von einem Anwender

VoIP Kurzform für Voice over IP, das Telefonieren über das Netzwerk/Internet

WinkeyNet Software, die die Steuerung und Konfigurierung von Boxen übernimmt

WMI Windows Management Instrumentation, ist ein Interface über das nahezu alle Systemeinstellungen ausgelesen und verändert werden können

WQL Windows Management Instrumentation Query Language, ein SQL-Dialekt, der von Microsoft entwickelt wurde und sich an Standard ANSI SQL orientiert

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 19. Oktober 2010